

POD 7 COPY

DO NOT REMOVE

SSTC Lumonics JK-701
Laser Welder Shared Beam
System Software Modules



Solid State Technology Center (SSTC)

Lumonics JK-701 Laser Welder

Shared Beam System Software Modules

POD 7 COPY

Do NOT Remove



Digitized by the Internet Archive
in 2025

https://archive.org/details/manualsplus_06878

CONTENTS

1. SHARED BEAM SYSTEM OVERVIEW	1
2. WELDING STATION SOFTWARE	2
2.1 SAMPLE WELDING PROCESS PROGRAM {TALKTOPC7}	2
2.2 COMMON SOFTWARE {WELD_SUBS}	2
2.3 WELDING STATION SOFTWARE CONCLUSION	6
3. HOST PC SOFTWARE	7
3.1 DESCRIPTION OF OPERATION	7
3.2 HOST PC SUBROUTINES & FUNCTIONS {LWELD11.BAS}	11
3.3 HOST PC SUBROUTINES & FUNCTIONS {MENU.BAS}	12
3.4 LIBRARY ROUTINES	13
4. HARDWARE CONFIGURATION	15
4.1 INMAC 8 PORT 'Any Port To Any Port' SMART SWITCH	15
4.2 RS-232 CABLE CONNECTIONS	15
4.3 RS-232 ADAPTER CABLES	16
4.4 Digital I/O and the Interface Box	17

CHAPTER 1

1.1. THE NATURE OF THE PROBLEM

The first step in the solution of a problem is to understand the nature of the problem. This involves a careful study of the problem statement and the identification of the key elements of the problem. The next step is to formulate a plan of attack, which involves deciding on the methods and techniques to be used in the solution of the problem.

1.2. THE FORMULATION OF THE PROBLEM

The second step in the solution of a problem is to formulate the problem. This involves identifying the key elements of the problem and the relationships between them. The next step is to formulate a plan of attack, which involves deciding on the methods and techniques to be used in the solution of the problem.

1.3. THE SOLUTION OF THE PROBLEM

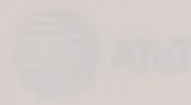
The third step in the solution of a problem is to solve the problem. This involves applying the methods and techniques identified in the plan of attack to the problem. The next step is to check the solution, which involves verifying that the solution is correct and that it satisfies the requirements of the problem.

LIST OF FIGURES

Figure 1. Sample Welding Station Menu	3
Figure 2. Custom Pulse Shape Menu	4
Figure 3. Host PC Menu	8
Figure 4. Example Error Message	9
Figure 5. Automatic Mode Display Screen	10

QUESTION 1

1. Which of the following is NOT a characteristic of a good research question?
2. Which of the following is NOT a characteristic of a good research question?
3. Which of the following is NOT a characteristic of a good research question?
4. Which of the following is NOT a characteristic of a good research question?
5. Which of the following is NOT a characteristic of a good research question?



LIST OF EXHIBITS

Exhibit 1. SAMPLE WELDING STATION PROGRAM: TALKTOPC7	19
Exhibit 2. COMMON WELDING STATION SOFTWARE: WELD_SUBS	20
Exhibit 3. HOST PC SOFTWARE: LWELD.BAS	21
Exhibit 4. HOST PC SOFTWARE: MENU.BAS	22
Exhibit 5. HOST PC SOFTWARE: MENU.INC	23
Exhibit 6. HOST PC SOFTWARE: DECLARE.INC	24
Exhibit 7. PROCESS DIAGRAMS	25
Exhibit 8. SAMPLE COMMUNICATION PRINTOUTS	26

TABLE 1

1	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029
2	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029
3	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029
4	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029
5	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029
6	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029
7	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029
8	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029
9	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029
10	1960-1969	1970-1979	1980-1989	1990-1999	2000-2009	2010-2019	2020-2029



AT&T Bell Laboratories

subject: **Solid State Technology Center (SSTC) Lumonics JK-701
Laser Welder Shared Beam System Software Modules**

date: **December 31, 1990**

from: **J. M. Guss
STC 52326
2C-253 x2777**

**B. D. Ehrhart
RD 2430
20-1-5B x3848**

PROGRAM DOCUMENTATION

1. SHARED BEAM SYSTEM OVERVIEW

The Lumonics JK-701 Laser Welder at SSTC is configured with a "Slow Time Share" beam switch and fiber optic delivery system. It is used to direct the laser's output to any 1 of 4 possible positions.

Each position is associated with a "Welding Station". The term "Welding Station" refers to the collection of hardware consisting of a welding enclosure, optics for focusing the laser beam, video cameras to view weld target, motorized Klinger submicron movement stages for positioning the parts to be welded, controllers for the motorized stages, bias and measurement circuitry (for the device to be welded) and an HP 9000/300 series computer. The Welding Station computer controls the movement of the stages and communicates with the Host PC, the Lumonics JK-701 Laser Welder and RAN via the "Smart Switch". The AT&T PC 6300 computer used to control the Lumonics JK-701 Laser Welder and fiber optics delivery system is referred to as the "Host PC". The computer system that supports the collection of data from this system is referred to as "RAN", the Reliability Analysis Network. The "Smart Switch" refers to the 8 port Inmac "Any Port To Any Port" RS-232 switching box.

In the present configuration, beam switch position #1 is subsequently split into 4 equal power beams and delivered to Welding Station #1. Beam switch position #2 is split into 2 equal beams and delivered to Welding Station #2. Beam switch position #3 is delivered via a single fiber to Welding Station #3. Presently beam switch position #4 is connected to an energy dump.

The interfaces and procedures for remote operation and digital control from the Welding Station(s) using a host computer are discussed in this document. All normal operations of the Welding Stations may be remotely controlled, except the following: Lumonics JK-701 Power Supply ON/OFF, air flow to Beam Switch Modules, the balancing of the laser beam via the Energy Share Modules.

2. WELDING STATION SOFTWARE

This memorandum documents the software written on the Welding Stations used for controlling the Lumonics shared beam laser welding system. The Welding Stations run programs written in HP-BASIC. A computer program to control the shared beam system is divided into two parts. The first part consists of a program file named WELD_SUBS that contains all of the subroutines that are needed for controlling the welder. These subroutines are used to communicate with the AT&T-6300 Host PC and the Lumonics JK-701 Laser Welder through the Smart Switch. The other program file described, TALKTOPC7, is a sample of a program that would be written to handle the needs of a given welding process. The names of the subroutines used in the program coding are contained below with a short explanation of each subroutine's function.

2.1 SAMPLE WELDING PROCESS PROGRAM {TALKTOPC7}

2.1.1 Main Routine of Welding Process Program The generic needs of the main program modules used in a laser welding process are as follows:

- First, the labeled common memory areas must be defined so that subroutines can easily share information known by the main program. (/Data_comm/ and /Pc_stuff/) Only the labeled areas containing variables needed by a subroutine must be included in that subroutine.
- Second, the main routine must load in the subroutines that are common to all programs that use the shared beam system. Before doing this, the program must assure that an older version of the common subroutines does not exist. The DELSUB and LOADSUB commands are used to accomplish these activities.
- Third, the main program needs to initialize the variables and buffers used in the transfer of data between processors by calling the Setup_serial subroutine.
- Last, the heart of the main program is a loop that allows the operator to select the options that are available from the given menu (see figure 1). The menu could include choices such as information entry, pre-defined welding subroutines, setup of pulse shape, target welding, printing of log files and the ability to exit this program and load the program menu available for the Welding Station.

2.1.2 Custom Subroutines The subroutines described below are examples of what the customized subroutines must contain in order for them to properly interact with the common subroutines.

- **Weld** - This subroutine should be given a unique name in the given welding program. The subroutine handles the setup needed for controlling the shared beam system (Pc_setup subroutine), calls the desired custom welding subroutine based on the parameter passed into it (in this example, the Multiple subroutine), and terminates the interaction with the shared beam system (Pc_signoff subroutine). This subroutine enables and disables the softkey used to activate the Operator_abort subroutine.
- **Multiple** - This sample subroutine is a simple loop to simulate a welding process where more than one weld is needed.

2.2 COMMON SOFTWARE {WELD_SUBS}

2.2.1 System Limitations The major concern with the shared beam system was how to really make it a shared system. The following limitations were designed into this system to prevent it from being dominated by one station or hung-up due to an anomaly in the communications.

- To promote sharing, a two minute maximum time usage limit for a given welding process was designed into the system.
- The Smart Switch manages all communications with a one minute timeout set. Therefore, it will disconnect a connection if there is no activity for that time period.

Figure 1. Sample Welding Station Menu

Sample Weld Menu 19 Oct 1990

M --> Multiple Shot Sample Weld

S --> Setup Pulse Shape

T --> Target Weld

<ESC> --> Exit to Program Menu

Select Desired Option [M,S,T or <ESC>] & press RETURN

					System	Caps	Command	
Step	Continue	RUN	Print All	Clr Tab Set Tab	Display Fctns	Any Char	Recall	*

Figure 2. Custom Pulse Shape Menu

CUSTOMIZE PULSE SHAPE MENU

19 Oct 1990

R--> Rep Rate : 2 Hz

W--> Width : 1 mS

H--> Height : 90 %

<ESC>--> Return to Main Menu

Select parameter to be changed [R,W,H or <ESC>] & press RETURN

				System	Caps	Command	
Step	Continue	RUN	Print All	Clr Tab	Display	Any	
				Set Tab	Fctns	Char	Recall *

2.2.2 Common Subroutines The following subroutines are common to all Welding Station programs used to communicate with the shared beam system. These subroutines must not be changed in any way without a co-ordinated effort to effect needed changes on the Host PC software.

- **Weld_subs** - A dummy subroutine used only to mark the beginning of the common welding subroutines. The DELSUB performed by the main program uses this subroutine as the starting point for deleting subroutines.
- **Customize_pulse** - Sets up a loop that allows the operator to select the pulse shape parameter to be modified from the given menu (see figure 2). Upon selection of a parameter, the present value will be displayed and the new value will be checked for validity when entered.
- **Enter_serial** - Receives information sent to the serial port by the Host PC, the Lumonics laser welder, or the database computer. The data is handled through an IO buffer and is delimited by the standard carriage-return/line-feed or the optional delimiter passed to the subroutine. The subroutine automatically handles and controls recovery from errors.
- **Operator_abort** - Allows the operator to confirm a requested (or generated) abort or select to proceed with the welding process. If the process is to be aborted, the subroutine will set the appropriate flags needed to terminate the welding process. This subroutine can be accessed on demand when the defined (f4) softkey is active and is pressed by the operator at the Welding Station.
- **Pc_release** - Is used to verify that there is no extraneous data waiting to be read at the serial port at the conclusion of the welding process. Failure to do this could cause the Host PC to wait for the Smart Switch timeout to occur before releasing the Host PC to acknowledge other welding requests.
- **Pc_setup** - Handles the communication process to the Host PC through the Smart Switch. The Welding Station will wait for an instruction to be sent from the Host PC through the serial port and then execute the necessary steps to satisfy that instruction. The following is a list of valid instructions that can be received from the Host PC:
 - [START] - Indicates that the Host PC is ready for the Welding Station's request. The Welding Station will transmit [REQUEST] followed by the time/date that indicates when this process was started. The subroutine will then wait for further instructions.
 - [SETUP] - The Host PC will send this after the Lumonics welder has been powered up and the beam has been switched to the Welding Station of interest. The Welding Station will then transmit [SET] to indicate to the Host PC that it should now disconnect from the Smart Switch clearing the way for the Welding Station to setup the welder (using the Setup_welder subroutine). After which, the Welding Station will mark the time for indicating the amount of time left for this welding process to complete its tasks before being forced to share the welder with the other Welding Stations. The subroutine will then wait for further instructions.
 - [READY] - The Host PC will send this after it has sensed that the Welding Station and welder are done communicating. The Host PC will start the clock for limiting the amount of time that this Welding Station may use the welder before it has to disconnect and share the system. The Welding Station will transmit [YES] if it is ready to proceed, and then wait for further instructions. If for any reason the Welding Station is not ready to process or gets the [READY] message from the Host PC out of sequence, the Welding Station will terminate this request by sending [CANCEL].
 - [BEGIN] - The Host PC will send this after it has received the [YES] from the Welding Station. At this point, the setup is complete and the required welding task may be performed.
 - [QUIT] - The Host PC will send this if for any reason the Welding Station has previously requested a [CANCEL] or if the Host PC is "unhappy". Upon receiving this, the Welding Station will terminate the present welding procedure.
 - >>> ERROR Message - The Host PC will send this if it has sensed a problem with the setting up of the welder. At this point, the Operator_abort subroutine will be invoked and the operator can either try and fix the problem or abort this welding procedure.

At any time during the setup, the operator can request that the procedure be terminated. When this occurs, the Welding Station will replace the appropriate response indicated above with [CANCEL] and then this procedure will terminate.

The instructions from the Host PC are retrieved through the Enter_serial subroutine using the default (CR/LF) delimiter.

- **Pc_signoff** - Sends out the message to indicate to the Host PC that this welding process is done.
- **Setup_serial** - Initializes the variables and buffers needed to manage the communications received through the RS-232 serial port.
- **Setup_welder** - Manages the communication from the welding station to the Lumonics welder through the Smart Switch. The Welding Station must get the Smart Switch's attention, request connection to the Lumonics welder (port #7) and setup the Welding Station serial port to Xon/Xoff, 7 bit, even parity before it can communicate with the welder.

The Welding Station will then send the welder all of the information needed to setup the required pulse shape. The Welding Station will verify that the pulse shape has been correctly set and then wait until the laser power has been stabilized for the define pulse shape. If the power is not setup within the predefined limits, the Welding Station will adjust the pulse height and go back and resend the pulse shape information. The test will only attempt to reset the power a limited number of times before giving up. If the welder can not be setup properly, the subroutine will return with a FAIL status so that the Pc_setup subroutine can terminate this welding procedure. Otherwise, an OK status will indicate that the welder was properly setup.

Upon completion of the welder setup, regardless of the success or failure, the serial port will be set back to its default power up state of 8 bit, no parity, no-op and the Welding Station will disconnect from the Smart Switch. Messages from the welder will be retrieved through the Enter_serial subroutine using the ASCII DC1 character (CHR\$(17), Control Q) as the delimiter.

- **Spot_weld** - Sends a request to the Host PC to perform a spot weld. The subroutine ensures that the shot has been fired before the Welding Station software continues on and moves the parts to the next weld position. If the weld fails, then the subroutine gives the operator three options: (1) re-try the weld, (2) skip over this weld position, and (3) abort the rest of this welding procedure. If an attempt is made to retry the weld and the Host PC does not respond, then this subroutine will attempt to re-establish communication with the Host PC so that the welding procedure can be continued. If the operator desired to terminate the welding procedure (Option 3), this subroutine will not request any more welds.
- **Wait_to_weld** - Keeps the Smart Switch from timing out while the operator is re-aligning parts to be welded. It achieves this by sending a message to the Host PC who in turn sends back a response thus generating activity to keep the Smart Switch from timing out.
- **End_wsups** - A dummy subroutine used only to mark the end of the common welding subroutines.

2.3 WELDING STATION SOFTWARE CONCLUSION

The software for the Welding Stations is designed so that it will take minimal effort to support multiple programs on multiple stations. The common subroutine file (WELD_SUBS) is the key ingredient in this design. Through the use of the DELSUB and the LOADSUB command, the common subroutine file can be modified and the changes will automatically be available to all of the programs that use the welder. This enables the Welding Station to be flexible and product/procedure independent.

3. HOST PC SOFTWARE

The Host PC oversees and controls the operations of the JK-701 Lumonics Laser Welder, the beam switch and coordinates the sharing of the beam between each Welding Station. The Host PC controls the welding process through two means, the first is through its RS-232 port connected to the Smart Switch, the second is through the 16 bit parallel I/O board SID-85RD2105 which is connected to the Interface Box. The Interface Box is connected to the PL805 CNC port on the Lumonics laser welder. The Interface Box is designed to allow automated and manual control of the beam switch as well as automated control of the JK-701 Laser Welder. The Host PC software was written in Microsoft Quick Basic Version 4.5 and compiled into an EXEcutable file names LWELD11.EXE. This software is user friendly and menu driven. A copy of this source code is included in the appendix.

3.1 DESCRIPTION OF OPERATION

When the Host PC is booted the AUTOEXEC.BAT program will load the LWELD11.EXE program and start its execution. Then the Host PC will display a menu (see figure 3) that allows the selection of several manual functions or entry into the automatic mode. The manual functions include: Ramping laser power up or down (toggle action), Setting the beam switch, Entering the AUTO mode, Setting the system date or time, and Returning to DOS. If an error occurs while attempting one of the manual functions, a message window will pop down explaining the problem and next action to take. An example of a pop down error message is shown in figure 4. This error occurred when trying to Ramp Up the Laser Power with the Lumonics control pendant in the local mode.

Upon entering the AUTO Mode the Host PC will display a screen similar to figure 5. This screen shows the status of the Welding Stations, the JK-701 Laser Welder, and the Interface box. In the AUTO Mode, the Host PC (via the Smart Switch) polls all Welding Stations to see if any are ready to weld. Each Welding Station, that is ready to weld, will be asked for its request time, the Host PC then prioritizes the request times and creates a welding queue. When the Host PC is ready to service a Welding Station it will send the message "[SETUP]" to the Welding Station's serial port (via the Smart Switch). The Host PC then disconnects from the Smart Switch to allow the Welding Station (through the switch) to connect to the JK-701 Laser Welder and setup the pulse shape as required. After the Welding Station has finished communications with the JK-701 Laser Welder, the Host PC and Welding Station re-establish communications. Thereafter, the Host PC starts a 2 minute time cycle. During the weld cycle, the Welding Station may request "[FIRE]"ing as many single shots as are necessary until the 2 minute time out occurs. Before the Host PC allows each single shot to be fired it checks the status of the laser, beam switch and door interlocks, if everything is ok a shot will be fired. If not, a warning message will be returned to the Welding Station. If the serial communications between the Welding Station and Host PC, RAN or JK-701 Laser Welder is interrupted for more than 1 minute the Smart Switch will automatically disconnect and the welding procedure will be aborted. During the welding procedure, the Welding Station can send a "[CANCEL]" to the Host PC at any time and stop welding. Once a Welding Station has concluded welding either through "[CANCEL]"ing or timing out, the Host PC will go back to check its weld queue and the next Welding Station will be allowed to weld. If no other Welding Stations are in the weld queue then the Host PC will go back to polling the Smart Switch to find the next Welding Station that is ready to weld. During the time when no Welding Stations are requesting service and the Host PC is polling the Smart Switch "Press END to Return to Manual Mode" message will appear at the bottom of the Host PC's CRT. If the END key is pressed during this time the Host PC will return it's MANUAL mode menu.

The software in the Host PC is an EXEcutable program that was created by compiling and linking two BASIC programs, two include files along with several 3rd party library modules and some code written in "C".

Figure 3. Host PC Menu

BELL LABS SSTC LASER WELDER HOST COMPUTER		
Version 2.20		11-27-90
Nov 27, 1990		1:14:10 pm

RAMP UP LASER POWER
BEAM Switch
AUTO Mode
Set Date
Set Time
Quit to DOS

• (Power Up Laser) [ENTER or L] to Select • [ESC] to Abort

Figure 4. Example Error Message

BELL LABS SSTC LASER WELDER HOST COMPUTER		
Version 2.20	11-27-90	
Nov 27, 1990		1:56:37 pm

RAMP UP LASER POWER BEAM Switch AUTO Mode Set Date Set Time Quit to DOS
--

WARNING! LASER IS NOT IN REMOTE! USE LASER CONTROL PENDANT AND SET LASER TO REMOTE!

Press [ENTER] to Try Again OR [ESC] to Abort

Figure 5. Automatic Mode Display Screen

BELL LABS SSTC LUMONICS JK 701 LASER STATUS		
Nov 27, 1990	AUTO MODE	1:14:25 pm

WORKSTATION STATUS				
QUE STATUS =>	WS-1 INACTIVE	WS-2 INACTIVE	WS-3 INACTIVE	WS-4 INACTIVE
DOOR INTERLOCK =>	OPEN	OPEN	OPEN	OPEN
JK-701 LASER STATUS				
AC POWER OFF	REMOTE YES	READY NO	POWER LIMIT OK	ENERGY TRIP OK
INTERFACE STATUS				
REMOTE NO	AIR PRESS. LOW	BEAM POS WS-4	SHUTTER	SINGLE SHOT

Press END to Return to Manual Mode

3.2 HOST PC SUBROUTINES & FUNCTIONS {LWELD11.BAS}

The following describes the BASIC sub programs and functions written for the Host PC.

- **AutoWeldMode** - Permits the HOST PC to control the Laser welding procedure in an automatic mode.
- **BgDelay** - Performs a background delay timing, returns control to the caller after each check so that the caller may do something else while waiting. Uses milliseconds and allows up to 10 delays to be running simultaneously. Resolution is limited to 54.945mS.
- **ClearDispVars** - Clears display variables so UpdateStatDisplay will work on re-entry.
- **CloseScreen** - Draws the closing screen and displays a message.
- **ConnectWs%** - Attempts up to 5 times to connect the Host PC to the Welding Station. Returns 1 for success, 0 for failure.
- **DispMsg** - Displays one variable length message, does not save or restore screen.
- **DispStat** - Displays the grid and headings that will show the status of the laser, interface box and queue. This grid will be filled with data when the UpdateStatDisplay subprogram is called (during automatic operation).
- **DispWarning%** - Displays Warning\$ message (up to 5 lines) and return ASCII value for either [ESC] or [ENTER] when one of these keys are pressed.
- **Filter** - Filters out backspace, line feed, carriage return, ctrl E, and ctrl F characters in the input string.
- **GetSerialData** - Check the serial port. If characters are available read them in, allows up to 10 seconds to receive characters.
- **HardDelay** - Performs a "hard" delay for a specified number of milli-seconds. NOTE: Resolution limited to 54.945mS.
- **InitWelder%** - This function assures a safe condition upon start up of the program. It will initialize the 16 bit interface which will set the beam switch to the dump shutter (position 4) and shut down all laser welder functions.
- **OktoWeld%** - Checks laser and interface box to see if its ok safely fire a single shot. Returns 0 if not OK, 1 if OK (no errors).
- **PortStatus%** - Attempts to read the status of Welding Station ports from the Smart Switch, it returns 1 for successful reading of switch or 0 for failure to read switch successfully. This function puts each Welding Station's status flag in PortStat array (0 if "NOT READY", 1 if "READY", 2 if "BUSY").
- **Pround!** - Returns the value of the Single precision argument rounded to a specified power of ten, as specified by the passed PowerOfTen rounding value.
- **PWR%** - Returns the power of 2 that the number Y is raised to.
- **RampLaser%** - Ramps laser power UP or DOWN (depending on direction specified), returns 1 for success or 0 for failure.
- **ReQue** - Finds the current Queue order based on the variable QueTime!() and stores the resultant Queue order in the QueStat() array. 2-4 = Queue position 1 = ready to weld, 5 = Canceled or Inactive.
- **SetupWs%** - Attempts to establish communication with Welding Station. Then informs Welding Station to setup welder through it's serial port. This function waits up to 60 seconds for the Welding Station to complete communication to the welder and become "READY" to communicate with the Host PC again. It returns 1 if communications are successfully reestablished or 0 if nor or if "CANCEL" is received.

- **SweepScreen** - Clears screen page 0, in a circular motion, to reveal the contents of screen page 1 (previously filled by whatever means is desirable).
- **SwitchBeam%** - Attempts to switch beam to desired Welding Station. It returns 0 if fail, 1 if successful.
- **Ticks!** - Returns the number of clock ticks since midnight. Very useful in timing routines.
- **UpdateStatDisp** - Determines and displays Laser, Interlock and Welding station Status on CRT
- **UpdateTime** - Displays Time & Date on the Status screen.
- **Weldit** - Dedicates up to 2 minutes to Welding station to do it's spot welds.
- **WsComm%** - Attempts to establish communication with Welding Station, and returns a passed in message to Welding Station if [CANCEL] is not received! This function will make up to 5 attempts to communicate. The time limit for data to be received is determined in Get Serial Data routine. It returns 1 if all is ok, 0 if [CANCEL] was received.

3.3 HOST PC SUBROUTINES & FUNCTIONS {MENU.BAS}

- **ClrScr** - Clears the screen in 2 different ways. The parameters are as follows:
 1. **mode%** - A number indicating which way you want the screen cleared. The number must be in the range of 0 through 14.
 2. **FillChar\$** - This is a single character string containing the character you want to clear the screen with. Under most circumstances, this will simply be a space.
- **DisplayEntry** - Is used only by the MakeMenu% Function. It is not meant for use on its own. The routine displays the passed menu entry on the screen, and highlights the character that proceeds the marker character.
- **ErrorTone** - Makes a nice but strange noise to signal an error condition.
- **MakeMenu%** - Displays a menu list on the screen and allows the user to move a scrolling selection bar to highlight the entry of their choice. Selection is made by pressing the <ENTER> key. Other allowable keys include <HOME> or <PG UP> to move to the first menu entry, and <END> or <PG DN> to move to the last entry. The scroll bar wraps from top to bottom and bottom to top. The function returns, as an Integer value, the position of the entry in the list of the user's selection. For example, if the user selected the third item in a list of eight, the function would return a value of three.
- **MakeWindow** - Draws windows on the screen. The kinds of windows you can make are quite varied. There are four window types, three frame types, windows can have shadows or not, you can "explode" them onto the screen, and even place labels on them.
- **OffCenter** - Centers the text passed to it on the screen between two specified columns. Excellent for centering text in a window that itself is not centered in the screen.
- **Oops** - Produces a pleasant tone instead of the dull BEEP offered on the PC.
- **ScreenBlank\$** - Blanks the screen to prevent burn-in and displays a message explaining this, until a key is hit. This function returns a string value for the key that was pressed.
- **ScreenSave** - Saves all or a portion of the screen display to an integer array.
- **ScrnRestore** - Restores all or a portion of the screen display from an integer array.
- **SetDate%** - Validates the entering of the correct string format for the date, and sets the CMOS RAM chip date for 286/386 machines. The DOS date is also updated immediately so this value can be used for the date display. SetDate% is returned as a 0 if invalid, or a -1 if valid. (Passed parameter StrDate\$: A string argument in the format mm-dd-yyyy, valid range of mm = 1 to 12; dd = 1 to 31; yyyy = 1980 to 1999.)

- **SetTime%** - Validates the entering of the correct string format for the time, and sets the CMOS RAM chip time for 286/386 machines. The DOS time is also updated immediately so this value can be used for the time display. SetTime% is returned as a 0 if invalid, or a -1 if valid. (Passed parameter StrTime\$: A string argument in the format hh-mm-ss, valid range of hh = 0 to 23 hrs; mm = 0 to 59 mins; ss = 0 to 59 secs.)

3.4 LIBRARY ROUTINES

3.4.1 Quick BASIC SERIAL I/O Version 1.2 LIBRARY ROUTINES

by Jeff Sumberg

Sysop(2) - SailBoard BBS [or] Box 212
Wayne, NJ, 201-831-8152 Ringwood, NJ, 07456

This library will provides you with serial I/O communications routines for use in Quick BASIC 4.x programs. In the spirit of programmer to programmer I am not requesting any compensation for the use of these routines other than you acknowledge in your documentation that you are using them. For more detailed explanation see the QB SERIAL.DOC file.

- **CarrierLost%** - A real time function, that returns the current carrier condition at the time of the call.
- **CarrierDetect** - Allows the calling program to control if the driver senses carrier. The driver defaults to carrier detection ON unless you implicitly turn it off. The following call has been added to do this: CarrierDetect 0 [or] 1. Specifying a ZERO turns carrier detection OFF. Even if you have the UEVENT turned ON, no trap will occur if carrier is lost. Also, any internal loops (such as XOFF wait) will NOT exit if carrier is lost. Specifying a ONE turns carrier detection back ON.
- **CDtrap** - Allows you to detect loss of carrier without using the UEVENT trap.
- **ClearInputBuffer** - Flushes input buffer of any existing characters.
- **CloseComm** - Restores interrupt vectors grabbed by OpenComm.
- **DataWaiting%** - Is used to test the input buffer to see if any characters need to be read from it.
- **DTRcontrol** - Allows control of the DTR line.
- **OpenComm** - Is a driver that provides serial port initialization.
- **ReadChar%** - Reads a single character from the buffer.
- **Transmit** - Sends a string of data out the serial port.
- **WriteChar%** - Writes a single character to the serial port.

3.4.2 Crescent Software 'QuickPak' Professional

- **ClearScr0** - Clears all or part of the screen to a specified color.
- **FillScrn** - Fills any rectangular portion of the display screen with the specified character.
- **MonthName\$** - Accepts an integer value between 1 and 12, and returns an equivalent 3 character month name as a string.
- **MPRestore** - Re-displays any rectangular portion from a screen that has been saved to an array using ScrnSave)0.
- **Qprint0** - Displays a string very quickly at the current cursor location.
- **ScrnSave** - Saves all or part of a text screen into an integer array, to allow restoring it again at a later time.

3.4.3 'C' Language Drivers Version 1.0 (01-12-90) B. Ehrhart {16BITBD.OBJ}

- **Enter16%** - Reads 16 bits of data from the digital input lines of the 16 bit I/O board at the specified address.

- **Output16%** - Writes 16 bits of data to the digital output lines of the 16 bit I/O board at the specified address.
- **Readio** - Reads data from a specified register of the 16 bit I/O board at the specified address.
- **SetTimeout** - Changes the default time to the specified time in mS before a handshake error will be detected during ENTER and OUTPUT operations.
- **Writeio** - Writes data to a specified register of the 16 bit I/O board at the specified address.

3.4.4 *BITS.OBJ* by Doug Mace

- **Bits%** - Is an alternative method of examining the status of one or more bits in an integer, by telling the function what bits are of interest. Saves you from figuring out bit masks.

3.4.5 *QB.LIB* by Microsoft

- **INTERRUPT** - Allows BASIC to make DOS system calls.

4. HARDWARE CONFIGURATION

4.1 INMAC 8 PORT 'Any Port To Any Port' SMART SWITCH

4.1.1 Individual RS-232 Smart Switch Port Descriptions

PORT	HANDLE	DEVICE	TIMEOUT	NOTES
1	HOST	AT&T 6300	NONE	DIRECT CABLE
2	WS-1	HP 310	1 Mintue	DIRECT CABLE
3	WS-2	HP 310	1 Minute	DIRECT CABLE
4	WS-3	HP 310	1 Minute	FIBER CONNECTION
5	WS-4	HP 310	1 Minute	RESERVED FOR WS-4
6	----	NONE	NONE	NOT USED PRESENTLY
7	JK701	JK-701 LASER	NONE	FIBER CONNECTION 9600 7 BIT EVEN PARITY
8	RAN	RAN SYSTEM	NONE	FIBER CONNECTION 9600 8 BIT NO PARITY

4.1.2 Other Settings (All Ports)

OPTION	SETTING
HANDSHAKE	NONE
PARITY	NONE (8 BIT NO PARITY or 7 BIT EVEN PARITY)

4.2 RS-232 CABLE CONNECTIONS

4.2.1 AT&T Fiber Modems to Smart Switch:

- Set Push Button To DCE (pressed In).
- Attach a NULL Modem to Smart Switch with the addition of a Short between Pin 5 and 20 (DTR - CTS).
- Connect the Fiber Modem to the NULL Modem. Note: Should also work without a NULL Modem if the short between pin 5 and 20 is made!

4.2.2 Copper Wire to Smart Switch from Host PC (AT&T) or Welding Station (HP):

- Set Push Button To DCE (pressed In).
- Connect cable directly to Smart Switch.

4.3 RS-232 ADAPTER CABLES

4.3.1 HP RS-232 9 pin to 25 pin Adapter Cable: The 9 pin RS-232 port on the HP-9000/300 Series Computer needs a cable with the following connections in order to connect to the 25 pin RS-232 port on the Smart Switch. This adaptor cable can be used with either cable or a fiber modem connection.

HP End 9 pin	Switch End 25 pin
-----------------	----------------------

1 ————— 20

2 ————— 2

3 ————— 3

4 ————— 4

5 ————— 5

6 ————— 6

7 ————— 7

8 ————— 8

9 ————— 22

4.3.2 HP RS-232 25 pin to 25 pin Adapter Cable: There are some older HP-9000/300 Series Computer that have a 25 pin RS-232 port and they need a cable with the following connections in order to connect to the 25 pin RS-232 port on the Smart Switch. This adaptor cable can also be used with either cable or a fiber modem connection.

HP End 25 pin	Switch End 25 pin
------------------	----------------------

2 ————— 2

3 ————— 3

4 ————— 4

5 ————— 5

6 ————— 6

7 ————— 7

8 ————— 8

20 ————— 20

4.4 Digital I/O and the Interface Box

4.4.1 DIGITAL INPUTS From INTERFACE BOX

Digital Input	Bit Value	Function Monitored
DI0	2 ⁰ (1)	Laser Sync 1=SYNC'd, 0=NOT SYNC'd
DI1	2 ¹ (2)	Laser Ready 1=YES, 0=NO
DI2	2 ² (4)	Laser Remote 1=REMOTE, 0=LOCAL
DI3	2 ³ (8)	Laser Power Limit 1=OK, 0=FAIL
DI4	2 ⁴ (16)	Laser Energy Trip 1=OK, 0=FAIL
DI5	2 ⁵ (32)	Door Interlock Welding station 1. 1=CLOSED, 0=OPEN
DI6	2 ⁶ (64)	Door Interlock Welding station 2. 1=CLOSED, 0=OPEN
DI7	2 ⁷ (128)	Door Interlock Welding station 3. 1=CLOSED, 0=OPEN
DI8	2 ⁸ (256)	Reserved Door Interlock Welding station 4. 1=CLOSED, 0=OPEN
DI9	2 ⁹ (512)	Interface Box 1=REMOTE, 0=LOCAL
DI10	2 ¹⁰ (1024)	Air Pressure to Beam Switch 1=OK, 0=LOW
DI11	2 ¹¹ (2048)	AC Power to Laser 1=ON, 0=OFF
DI12	2 ¹² (4096)	Unused
DI13	2 ¹³ (8192)	Unused
DI14	2 ¹⁴ (16384)	Unused
DI15	2 ¹⁵ (32768)	Unused

4.4.2 DIGITAL OUTPUTS To INTERFACE BOX

Digital Output	Bit Value	Function Controlled
DO0	2 ⁰ (1)	Unused
DO1	2 ¹ (2)	Unused
DO2	2 ² (4)	Unused
DO3	2 ³ (8)	Ramp Laser 1=UP, 0=DOWN
DO4	2 ⁴ (16)	Shutter 1=OPEN, 0=CLOSED
DO5	2 ⁵ (32)	Single Shot (Toggle 0,1,0 ~ 200mS)
DO6	2 ⁶ (64)	Range Select Unused Leave 0
DO7	2 ⁷ (128)	Beam Switch (See table below)
DO8	2 ⁸ (256)	Beam Switch (See table below)
DO9	2 ⁹ (512)	Unused
DO10	2 ¹⁰ (1024)	Unused
DO11	2 ¹¹ (2048)	Unused
DO12	2 ¹² (4096)	Unused
DO13	2 ¹³ (8192)	Unused
DO14	2 ¹⁴ (16384)	Unused
DO15	2 ¹⁵ (32768)	Unused

4.4.2.1 BEAM SWITCH TABLE

Bit DO8	Bit DO7	Combined Value	Beam Position Welding Station #
0	0	(0)	4
0	1	(128)	1
1	0	(256)	2
1	1	(384)	3

Exhibit 1. SAMPLE WELDING STATION PROGRAM: TALKTOPC7


```

10! RE-STORE "TALKTOPC7"
20! [NOTE: Before storing, execute the following line:
30! DEL Weld_subs,End_wsubs
40! Sample Program for Laser Welder Control. J.M.GUSS SSTCx2777 11/19/90
50! Lumonics Shared beam laser welding system. [Test Set <-> PC(B.D.Ehrhart)]
60 COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_seria
1
70 COM /Pc_stuff/ Message$[80],Setup$[25],Start$[25],Fire$[10],Rep_rate,Pwidth
h,Pheight,Weld_started,Target_powerJoules,Target_window,
80 ON ERROR GOTO Load_subs
90 DELSUB Weld_subs TO END
100 Load_subs:OFF ERROR
110 LOADSUB ALL FROM "WELD_SUBS"
120 CALL Setup_serial
130! "SP1,2.0Hz,1.0mS,90.0%" <- sample of Pulse Shape string sent to welder.
140 Rep_rate=2.0 !Hz
150 Pwidth=1.0 !mS
160 Pheight=90.0 !%
170 Target_powerJoules=7.5 !joules
180 Target_window=1 !joule (+/-)
190 LOOP
200 Z$=""
210 PRINTER IS CRT
220 PRINT CHR$(12);
230 PRINT TABXY(20,2);"Sample Weld Menu";TAB(65);DATE$(TIMEDATE)
240 PRINT
250 PRINT TABXY(10,4);CHR$(133);" M ";CHR$(128);"--> Multiple Shot Sample We
ld"
260 PRINT TABXY(10,6);CHR$(133);" S ";CHR$(128);"--> Setup Pulse Shape"
270 PRINT TABXY(10,7);CHR$(133);" T ";CHR$(128);"--> Target Weld"
280 PRINT TABXY(9,9);CHR$(133);"<ESC>";CHR$(128);"--> Exit to Program Menu"
290 INPUT "Select Desired Option [M,S,T or <ESC>] & press RETURN",Z$
300 SELECT Z$
310 CASE "M"
320 Stime=TIMEDATE
330 Rep_rate=1.0 !Hz
340 Pwidth=3.0 !mS
350 Pheight=95.0 !%
360 Target_power=24
370 Target_window=1.5
380 CALL Weld("M")
390 Ftime=TIMEDATE
400 PRINT "Minutes: ";(Ftime-Stime)/60,"Seconds: ";Ftime-Stime
410 INPUT "Press RETURN to go back to menu:",Z$
420 CASE "S"
430 CALL Customize_pulse
440 CASE "T"
450 Stime=TIMEDATE
460 CALL Weld("T")
470 Ftime=TIMEDATE
480 PRINT "Minutes: ";(Ftime-Stime)/60,"Seconds: ";Ftime-Stime
490 INPUT "Press RETURN to go back to menu:",Z$
500 CASE CHR$(27)
510 LOAD "MENU",1
520 END SELECT
530 END LOOP
540 END
550 !
560 Weld:SUB Weld(Type$)
570 COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
580 ON KEY 4 LABEL "ABORT" CALL Operator_abort
590 USER 1 KEYS
600 KEY LABELS ON
610 CALL Pc_setup
620 IF Status$[1,6]<>"[QUIT]" THEN

```



```

630     SELECT Type$
640     CASE "M"
650         CALL Multiple
660     CASE "T"
670         CALL Spot_weld(1)
680     END SELECT
690     CALL Pc_signoff
700     END IF
710     CALL Pc_release
720     SUBEND !Weld
730 !
740 Multiple:SUB Multiple
750     COM /Data_comm/ Text$(1024) BUFFER,@Buff,@Dev,Status$(80),INTEGER Sc_ser
ial
760     CALL Spot_weld(1)
770     IF Status$(1,4)<>"[OK]" THEN SUBEXIT
780     CALL Wait_to_weld !Keep communications active so operator can re-align.
790     FOR I=1 TO 10
800         WAIT .5 !Simulate movement of part!
810         CALL Spot_weld(I+1)
820         IF Status$(1,4)<>"[OK]" THEN SUBEXIT
830     NEXT I
840     SUBEND !Multiple
850 !
860 End_of_prog:!Any new subs must be added before this line! Before storing a
new version, all lines after this must first be deleted!
870 !

```


Exhibit 2. COMMON WELDING STATION SOFTWARE: WELD__SUBS


```

10! RE-STORE"WELD_SUBS"
20! Laser Welder Control Subroutines J.M.GUSS SSTCx2777 11/19/90
30! Lumonics Shared beam laser welding system. [Test Set <-> PC(B.D.Ehrhart)]
40 COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_serial
50 COM /Pc_stuff/ Message$[80],Setup$[25],Start$[25],Fire$[10],Rep_rate,Pwidth,
Pheight,Weld_started,Target_power,Target_window
60 !
70 Weld_subs:SUB Weld_subs !Dummy subroutine to mark start of common Lumonics
shared beam laser welding system subroutines.
80 SUBEND !Weld_subs
90 !
100 Customize_pulse:SUB Customize_pulse
110 COM /Pc_stuff/ Message$[80],Setup$[25],Start$[25],Fire$[10],Rep_rate,Pwidth,
Pheight,Weld_started,Target_power,Target_window
120 LOOP
130 Z$=""
140 PRINT CHR$(12);
150 PRINT TABXY(20,2);"CUSTOMIZE PULSE SHAPE MENU";TAB(65);DATE$(TIMEDATE)
160 PRINT TABXY(10,5);CHR$(133);" R ";CHR$(128);"--> Rep Rate : ";Rep_rate;
e;" Hz"
170 PRINT TABXY(10,7);CHR$(133);" W ";CHR$(128);"--> Width : ";Pwidth;
" mS"
180 PRINT TABXY(10,9);CHR$(133);" H ";CHR$(128);"--> Height : ";Pheight;
;" %"
190 PRINT TABXY(9,11);CHR$(133);"<ESC>";CHR$(128);"--> Return to Main Menu
"
200 INPUT "Select parameter to be changed [R,W,H or <ESC>] & press RETURN"
,Z$
210 SELECT Z$
220 CASE "R"
230 OUTPUT KBD USING "#,K";Rep_rate
240 INPUT "Enter Rep Rate (Hz):",Rep_rate
250 IF Rep_rate<=0 OR Rep_rate>15 THEN 240
260 CASE "W"
270 OUTPUT KBD USING "#,K";Pwidth
280 INPUT "Enter pulse Width (mS):",Pwidth
290 IF Pwidth<=0 OR Pwidth>15 THEN 280
300 CASE "H"
310 OUTPUT KBD USING "#,K";Pheight
320 INPUT "Enter pulse Height (%):",Pheight
330 IF Pheight<.1 OR Pheight>99.9 THEN 320
340 CASE CHR$(27)
350 SUBEXIT
360 END SELECT
370 END LOOP
380 SUBEND !Customize_pulse
390 !
400 Enter_serial:SUB Enter_serial(OPTIONAL Delimiter$) !Get data through serial
port.
410 COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
420 ON TIMEOUT Sc_serial,20 GOTO Serial_timeout
430 ON ERROR GOTO No_go
440 IF NPAR=0 THEN
450 TRANSFER @Dev TO @Buff
460 ELSE
470 TRANSFER @Dev TO @Buff;DELIM Delimiter$
480 END IF
490 ENTER @Buff;Status$
500 !!! PRINT "In Enter_serial: ";Status$
510 WAIT .01 !DANGER:Do NOT Remove (needed for receiving complete buffer!?)
520 ABORTIO @Dev
530 SUBEXIT
540 No_go:Status$="[AB]"
550 IF ERRN=167 THEN

```



```

560     STATUS Sc_serial,10;Uart_error
570     IF BIT(Uart_error,1) THEN PRINT "Overrun:";
580     IF BIT(Uart_error,2) THEN PRINT "Parity:";
590     IF BIT(Uart_error,3) THEN PRINT "Framing:";
600     IF BIT(Uart_error,4) THEN PRINT "Break:";
610     END IF
620     PRINT "Enter_serial:";ERRM$
630     SUBEXIT
640 Serial_timeout:PRINT "No response from Serial Port! (Timed Out)"
650     Status$="[T0]"
660     SUBEND !Enter_serial
670 !
680 Operator_abort:SUB Operator_abort
690     COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
700     COM /Pc_stuff/ Message$[80],Setup$[25],Start$[25],Fire$[10],Rep_rate,Pwi
dth,Pheight,Weld_started,Target_power,Target_window
710     BEEP
720     PRINT TABXY(1,19);"Abort requested by Operator! << Must respond within 4
5 sec.>>>"
730     INPUT "Enter A-Abort or C-Continue operation:",Z$
740     SELECT Z$
750     CASE "A"
760         Message$="[CANCEL]"
770         Setup$="[CANCEL]"
780         Start$="[CANCEL]"
790         Fire$="[DONE]"
800         Status$="[QUIT]"
810     CASE "C"
820         PRINT TABXY(1,19);"
"
830     CASE ELSE
840         GOTO 730
850     END SELECT
860     SUBEND!Operator_abort
870 !
880 Pc_release:SUB Pc_release !Let the PC disconnect from the switch.
890     COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
900     ON TIMEOUT Sc_serial,1 GOTO Release_timeout
910     CALL Enter_serial
920     PRINT "Pc_release:";Status$
930 Release_timeout:OFF TIMEOUT Sc_serial
940     SUBEND !Pc_release
950 !
960 Pc_setup:SUB Pc_setup !Startup Test Set to PC Communications.
970     COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
980     COM /Pc_stuff/ Message$[80],Setup$[25],Start$[25],Fire$[10],Rep_rate,Pwi
dth,Pheight,Weld_started,Target_power,Target_window
990     CONTROL Sc_serial,14;3 !Reset to Power up: 8 Bits/char, No Parity, Hands
hake No-op
1000    CONTROL Sc_serial,0;1 !Reset RS-232 Serial Interface
1010    Weld_started=TIMEDATE
1020    Message$="[REQUEST] "&VAL$(Weld_started)
1030    Setup$="[SET]"
1040    Start$="[YES]"
1050    Fire$="[FIRE]"
1060    PRINT CHR$(12);TABXY(15,1);"Started Communication w/PC @ ";TIME$(Weld_st
arted)
1070    Sequence=0
1080    Weld_started=0
1090    LOOP
1100 Restart:CALL Enter_serial
1110    PRINT "From PC: <<< ";Status$;" >>>";
1120    IF Weld_started>0 THEN

```



```

1130      PRINT TAB(45);" (Welding time left:";PROUND(2-(TIMEDATE-Weld_started
)/60,-2);" minutes)"
1140      ELSE
1150      PRINT
1160      END IF
1170      SELECT Status$[1,4]
1180      CASE "[STA"!RT]
1190      PRINT "Sending to PC: <<< ";Message$;" >>>"
1200      OUTPUT Sc_serial;Message$
1210      CALL Enter_serial
1220      PRINT "Got Back:";Status$
1230      Sequence=1
1240      CASE "[SET"!UP]
1250      PRINT "Setup welder: <<< ";Setup$;" >>>"
1260      OUTPUT Sc_serial;Setup$
1270      IF Setup$="[SET]" THEN
1280      CALL Enter_serial
1290      PRINT "PC disconnected - Ready to setup Welder! (";Status$;")"
1300      CALL Setup_welder
1310      IF Status$<>"[OK]" THEN Start$="[CANCEL]"
1320      Sequence=2
1330      Weld_started=TIMEDATE !PC starts timing just before [READY] sent!
1340      END IF
1350      CASE "[REA"!DY]
1360      IF Sequence<2 THEN Start$="[CANCEL]"
1370      OUTPUT Sc_serial;Start$
1380      PRINT "Ready to weld: <<< ";Start$;" >>>"
1390      Sequence=3
1400      CASE "[BEG"!IN]
1410      SUBEXIT
1420      CASE ">>> "!ERROR
1430      PRINT
1440      CALL Operator_abort
1450      CASE "[QUI"!T]!cancelled
1460      PRINT "Communications Cancelled!"
1470      CASE "[TO]"
1480      PRINT "PC did NOT respond (Timed Out). Welder probably being use by
another station."
1490      CASE "[RET"!RY]"
1500      CASE "" !Received a blank line (CR/LF)!
1510      CASE ELSE
1520      PRINT "Bad Message - Wait for another!"
1530      END SELECT
1540      EXIT IF Status$="[QUIT]"
1550      END LOOP
1560      SUBEND !Pc_setup
1570 !
1580 Pc_signoff:SUB Pc_signoff !Terminate the Test Set to PC Communications.
1590 COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
1600 ON TIMEOUT Sc_serial,10 GOTO Signoff_timeout
1610 OFF KEY 4
1620 PRINT "Signoff Sending: [DONE]"
1630 OUTPUT Sc_serial;"[DONE] "
1640 CALL Enter_serial
1650 PRINT "Signoff got from PC:";Status$;" --> ";
1660 SELECT Status$[1,5]
1670 CASE "[O&O]"
1680 PRINT "Communications Terminating!"
1690 CASE ELSE
1700 PRINT "Bad Message!?!?!"
1710 END SELECT
1720 SUBEXIT
1730 Signoff_timeout:PRINT "Signoff to PC NOT Responding! (Timed Out)"
1740 Status$="[QUIT]"
1750 SUBEND !Pc_signoff

```



```

1760 !
1770 Setup_serial:SUB Setup_serial
1780     COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
1790     Sc_serial=9
1800     ASSIGN @Buff TO BUFFER Text$
1810     ASSIGN @Dev TO Sc_serial
1820     SUBEND !Setup_serial
1830 !
1840 Setup_welder:SUB Setup_welder
1850     COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
1860     COM /Pc_stuff/ Message$[80],Setup$[25],Start$[25],Fire$[10],Rep_rate,Pwi
dth,Pheight,Weld_started,Target_power,Target_window
1870     DIM Command$[40]
1880     PRINT "About to wake-up Smart Switch!?"
1890     ON TIMEOUT Sc_serial,10 GOTO Welder_off
1900     OUTPUT Sc_serial USING "+,K";CHR$(5)
1910     PRINT "Requesting Connection to Port 7"
1920     OUTPUT Sc_serial USING "+,K";"C 7"
1930     CONTROL Sc_serial,0;1
1940     CONTROL Sc_serial,14;IVAL("01011010",2)!Xon/Xoff, EVEN, Enable Parity, 7
Bits
1950     FOR I=0 TO 14
1960         STATUS Sc_serial,I;A
1970         PRINT "(";I;")";A,
1980     NEXT I
1990     PRINT
2000     Count=0
2010     Limit=3
2020     REPEAT
2030         OUTPUT Sc_serial USING "+,K";"EW"
2040         CALL Enter_serial(CHR$(17))
2050         Count=Count+1
2060         UNTIL Status$[1,4]="WS00" OR Count=Limit
2070         IF Status$[1,4]<>"WS00" THEN Setup_failed
2080         Retry=99
2090         Attempt=0
2100         Adjust_height=2
2110 Reset_power:Count=0
2120         Attempt=Attempt+1
2130         OUTPUT R$ USING "#,ZZ.D";Rep_rate
2140         IF LEN(R$)=4 AND R$[1,1]="0" THEN R$=R$[2,4]
2150         OUTPUT W$ USING "#,ZZ.D";Pwidth
2160         IF LEN(W$)=4 AND W$[1,1]="0" THEN W$=W$[2,4]
2170         OUTPUT H$ USING "#,ZZ.D";Pheight
2180         IF LEN(H$)=4 AND H$[1,1]="0" THEN H$=H$[2,4]
2190         Command$="SP1,"&R$&"Hz,"&W$&"mS,"&H$&"%"
2200         OUTPUT Sc_serial USING "+,K";Command$
2210         Clen=LEN(Command$)
2220         IF Clen<=3 THEN GOTO Setup_failed
2230         PRINT "Setting: ";Command$,"Length: ",Clen
2240         REPEAT
2250             OUTPUT Sc_serial USING "+,K";"EX1"
2260             CALL Enter_serial(CHR$(17))
2270             Count=Count+1
2280             IF LEN(Status$)<3 THEN Status$="NFG"
2290             PRINT "Welder returned: ";Status$
2300             UNTIL Status$[3,Clen]=Command$[3,Clen] OR Count=Limit
2310             IF Status$[3,Clen]<>Command$[3,Clen] THEN Setup_failed
2320             Joules=-999
2330             Count=0
2340             REPEAT
2350                 Last=Joules
2360                 WAIT 1
2370                 OUTPUT Sc_serial USING "+,K";"EE1"

```



```

2380 CALL Enter_serial<CHR$(17)>
2390 IF LEN(Status$)>4 THEN
2400     Spos=POS(Status$,"")
2410     Jpos=POS(Status$,"J")
2420     IF Spos<Jpos AND Jpos>4 THEN Joules=VAL(Status$[Spos+1,Jpos-1])
2430 END IF
2440 Count=Count+1
2450 PRINT "Laser Power: ";Joules,Last,Count
2460 UNTIL ABS(Joules-Last)<.2 OR Count>20
2470 IF ABS(Joules-Target_power)>Target_window THEN
2480     IF Attempt<5 THEN
2490         IF Joules>Target_power THEN
2500             Adjust_dir=-1
2510         ELSE
2520             Adjust_dir=1
2530         END IF
2540         PRINT Attempt,"Re-adjusting Power from";Joules;"joules to ";Target_
power;" (+/-";Target_window;"")
2550         PRINT "Pulse Height was ";Pheight;"% changing to";
2560         Pheight=Pheight+(Adjust_height*Adjust_dir)
2570         IF Pheight>99.9 THEN Pheight=99.9
2580         IF Pheight<.1 THEN Pheight=.1
2590         PRINT Pheight;"%"
2600         GOTO Reset_power
2610     ELSE
2620         GOTO Setup_failed
2630     END IF
2640 END IF
2650 Count=0
2660 OUTPUT Sc_serial USING "+,K";"S01"
2670 PRINT "Select Active Pulse!"
2680 REPEAT
2690     OUTPUT Sc_serial USING "+,K";"EO"
2700     CALL Enter_serial<CHR$(17)>
2710     Count=Count+1
2720     IF LEN(Status$)<3 THEN Status$="NFG"
2730     UNTIL Status$[1,3]="PS1" OR Count=Limit
2740     IF Status$[1,3]<>"PS1" THEN Setup_failed
2750     Status$="[OK]"
2760 Reset_port:CONTROL Sc_serial,14;3!Reset to Power up: 8 Bits/char, No Parity
, Handshake No-op
2770     CONTROL Sc_serial,0;1
2780     OUTPUT Sc_serial USING "+,K";CHR$(13)&"+++"
2790     Retry=Retry+1
2800     IF Retry<9 THEN Setup_welder
2810     PRINT "Setup Welder Status: ";Status$
2820     OFF TIMEOUT Sc_serial
2830     SUBEXIT
2840 Setup_failed:Status$="[FAIL]"
2850     GOTO Reset_port
2860 Welder_off:BEEP
2870     PRINT TABXY(1,19);"Welder is NOT Powered up!"
2880     INPUT "Press RETURN when Welder is turned ON!";Z$
2890     PRINT TABXY(1,19);" "
2900     GOTO Setup_welder
2910 SUBEND !Setup_welder
2920 !
2930 Spot_weld:SUB Spot_weld(Number)
2940     COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
2950     COM /Pc_stuff/ Message$[80],Setup$[25],Start$[25],Fire$[10],Rep_rate,Pwi
dth,Pheight,Weld_started,Target_power,Target_window
2960     Tries=Tries+1
2970     WAIT .5
2980     PRINT "Asking for spot weld.";TAB(45);"<Welding time left:";PROUND(2-<TI
MEDATE-Weld_started)/60,-2);" minutes)"

```



```

2990 IF Fire$<>"[FIRE]" THEN SUBEXIT
3000 OUTPUT Sc_serial;Fire$
3010 CALL Enter_serial
3020 PRINT "Spot Weld # ";Number;" Status:";Status$;" --> ";
3030 SELECT Status$[1,4]
3040 CASE "[OK]"
3050 PRINT "Shot Fired!"
3060 WAIT .8/Rep_rate !Delay to make sure shutter opens before move!
3070 CASE "[QUI]"!T
3080 !Operator Aborted!
3090 CASE ELSE
3100 IF Tries>1 AND Status$[1,4]<>">>> " THEN
3110 Resetup:CALL Pc_setup
3120 IF Status$[1,6]<>"[QUIT]" THEN
3130 Tries=0
3140 GOTO Spot_weld
3150 ELSE
3160 PRINT "Could NOT reconnect to continue welding!"
3170 INPUT "Enter R-Retry or A-Abort: ",Z$
3180 SELECT Z$
3190 CASE "R"
3200 GOTO Resetup
3210 CASE "A"
3220 SUBEXIT
3230 CASE ELSE
3240 GOTO 3170
3250 END SELECT
3260 END IF
3270 ELSE
3280 PRINT "Welder indicated that it could not Fire!?"
3290 PRINT " Retry : Askes for another spot weld without moving."
3300 PRINT " Skip : Skips over this weld position and moves on to next
spot"
3310 PRINT " Abort : Terminates welding."
3320 INPUT "Enter R-Retry, S-Skip or A-Abort: ",Z$
3330 SELECT Z$
3340 CASE "R"
3350 GOTO Spot_weld
3360 CASE "S"
3370 Status$="[OK]"
3380 SUBEXIT
3390 CASE "A"
3400 Fire$="[DONE]"
3410 SUBEXIT
3420 CASE ELSE
3430 GOTO 3320
3440 END SELECT
3450 END IF
3460 END SELECT
3470 SUBEND !Spot_weld
3480 !
3490 Wait_to_weld:SUB Wait_to_weld
3500 COM /Data_comm/ Text$[1024] BUFFER,@Buff,@Dev,Status$[80],INTEGER Sc_ser
ial
3510 COM /Pc_stuff/ Message$[80],Setup$[25],Start$[25],Fire$[10],Rep_rate,Pwi
dth,Pheight,Weld_started,Target_power,Target_window
3520 PRINT
3530 PRINT TABXY(1,19);"Press RETURN when ready to continue welding:"
3540 ON KBD GOTO Input
3550 LOOP
3560 IF Fire$<>"[FIRE]" THEN SUBEXIT
3570 OUTPUT Sc_serial;"[WAIT]"
3580 CALL Enter_serial
3590 SELECT Status$[1,4]
3600 CASE "[OK]"
3610 CASE ELSE

```



```

3620         PRINT "Unexpected reply: ";Status$
3630     END SELECT
3640     FOR I=1 TO 10 !Talk about every 10 seconds.
3650         IF Fire$<>"[FIRE]" THEN SUBEXIT
3660         DISP "<<< Welding time left: ";PROUND(2-(TIMEDATE-Weld_started)/60,-2
);" minutes >>>";TAB(50);Status$
3670         WAIT .9
3680         Status$=""
3690     NEXT I
3700     END LOOP
3710 Input:PRINT TABXY(1,19);"Ready to resume welding!" !For now any input is OK
3720     SUBEND !Wait_to_weld
3730 !
3740     SUB End_wsubs !Dummy subroutine to mark end of common Lumonics shared beam
laser welding system subroutines.
3750 End_wsubs:SUBEND !<<< WARNING: Do NOT add lines beyond this point! >>>

```


Exhibit 3. HOST PC SOFTWARE: LWELD.BAS


```

DEFINT A-Z          ' All Variables as integers
DECLARE FUNCTION Pround! (N!, PowerOfTen%)
DECLARE FUNCTION OktoWeld% (WorkStation%)
DECLARE FUNCTION ConnectWs% (Ws%)
DECLARE FUNCTION SetupWs% (Ws%)
DECLARE FUNCTION WsComm% (Ws%, WsMessage$)
DECLARE SUB ClearDispVars ()
DECLARE SUB Harddelay (Milliseconds!)

REM $INCLUDE: 'DECLARE.INC'          ' Declare all Procedures
!*****
!
!               LWELD.BAS
!
! LWELD.BAS is a compiled Microsoft(tm) QuickBASIC 4.5 Main Program Module
! HOST control program for use with the HP SERIES 310 Laser Welding
! workstations (up to 4) and the BEAM SWITCH, SMARTSWITCH, LUMONICS JK-701
! at Bell Labs - SSTC
!
!
!           Written by B.D.Ehrhart Dept 2010 - MAY 11, 1990
!           AT&T Microelectronics, Reading, PA
! AT&T Microelectronics - Proprietary. Use Pursuant to Company Instructions
!*****
! Revision History:
!   August 21, 1990 LWELD.BAS Version 1.0      - Initial release
!   September 4, 1990 LWELD.BAS Version 2.0    - MANY MODS LATER
!   October 17, 1990 LWELD.BAS Version 2.1     - replaced Pause3 with
!           Harddelay and added Laser Power indicator to Auto mode
!           - B.D.EHRHART
!   November 19, 1990 LWELD11.BAS Version 2.11 - added WAIT as valid choice
!           to Weldit SUB - J.M.GUSS
!   November 27, 1990 LWELD11.BAS Version 2.20 - deleted "Future" choices
!           from opening screen menu - B.D.EHRHART
!-----
! -*- The Following Line Is Required To Show VERSION and DATE -*-
VersionStr$ = "Version 2.20          11-27-90"
!-----
!
! Finally, the author wishes to acknowledge the efforts of Doug Mace for the
! Screen Menu Graphics procedures and other invaluable help that is utilized
! in this program.
!-----
! QuickBASIC Environment Loading Command: QB /LLWELDER5[.QLB] LWELD[.BAS]
!
! The following files must be in the current directory when running from the
! environment:
!   LWELD.MAK      - the MAKE file required to load all modules into the QB
!                   environment.
!   LWELD.BAS      - Main Program Module
!   LWELDER5.QLB   - Quick Library of precompiled procedures for use within
!                   the QB development environment. Includes QBSERIAL, WINDOW,
!                   and 16BITBD.C, QB
!   DECLARE.INC    - INCLUDE file containing the procedure declarations for all
!                   modules.
!   MENU.INC       - INCLUDE file for initializing MENU.BAS module.
!   MENU.BAS       - Module containing the procedures used for drawing the
!                   operator menu screens.
!
!---- Additional files required for compile of LWELD.EXE:
!   LWELDER5.LIB   - Library of precompiled procedures searched by LINK when
!                   compiling the program to Stand-Alone EXEcutable form.
!                   This is a parallel to the .QLB Quick Library.

```



```

'
' MLIBCE.LIB - Medium Memory-Model Library for Microsoft QuickC - Ver 2.0
'
' NOTE: LWELD.BAS - MUST be compiled from the DOS command line to avoid pulling
'       in unnecessary procedures from the libraries.
' Compile Syntax = BC Lweld /O /V;
'               BC Menu /O /V;
'
' Link Syntax = LINK Lweld+Menu+NOCOM /NOE/E,LWeld,LISTFILE,BCOM45+LWELDER5
'
'---- External files required to run LWELD.EXE, these must be in the current
'       directory or DOS path:
'/=====/'
DIM SHARED Mode$          ' AUTO or MANUAL mode STORED HERE
Mode$ = "MANUAL"
DIM SHARED PortStat(8) AS INTEGER
COLOR 0, 1               ' Color background of screen
ClrScr 1, ""              ' Use 176,177,or 178 for different hues

'+++ Initialization Sequence Screen - Draw windows

'---- Initialize Variables for Windows
TRow = 2: BRow = 4: LCol = 10: RCol = 70
fg = 15: bg = 3           ' Fore/Background colors

'---- Draw Window - Initialize 16 bit Interface and Interface box
MakeWindow TRow, LCol, BRow, RCol, fg, bg, 0, 3, 0, 1, ""
OffCenter "Initializing 16 Bit Interface and Welder Interface Box", TRow + 1, LCol, RCol, 0, bg
IF InitWelder = FALSE THEN Oops ELSE OffCenter "Initialization OK!", TRow + 3, LCol, RCol, 0, bg: Harddelay 500 ' 1/2
SECOND WAIT

REM $INCLUDE: 'menu.inc'    ' Draw menu screen

'---- Act according to users Menu Choice
SELECT CASE Choice
CASE 1      '----- Power UP or DOWN Laser
    SELECT CASE Bits%(IboxControlWord, PWR(RampUp), PWR(RampUp))
    CASE 0      ' if NOT powered UP
        IF Ramplaser("UP") = 1 THEN
            ' if success switch the label for the menu
            Choices$(1) = "RAMP DOWN `LASER POWER"
            HelpText$(1) = " = (Power Down Laser) = [ENTER or L] to Select = [ESC] to Abort"
        END IF
    CASE 1      ' if powered UP
        IF Ramplaser("DOWN") = 1 THEN
            ' if success switch the label for the menu
            Choices$(1) = "RAMP UP `LASER POWER"
            HelpText$(1) = " = (Power Up Laser) [ENTER or L] to Select = [ESC] to Abort"
        END IF
    END SELECT
    CurrentLocation = 1          ' Return with menu choice 7 highlighted
    GOTO DrawScreen

CASE 2      '----- Select WorkStation SWITCH BEAM
    REDIM Ws$(4): Ws$(1) = "WS-1": Ws$(2) = "WS-2": Ws$(3) = "WS-3": Ws$(4) = "WS-4"
    TRow = 19: BRow = 24: LCol = 23: RCol = 57
    FColr = 0: BColr = 3: Label$ = " *- SELECT WORKSTATION *- "
    GOSUB OpenScr1
    Tr = 20: Br = 22: Lc = 27: Cw = 5: Inc = 7: Sfg = 14: Sbg = 4
    'MakeWindow Tr, Lc + 0 * Inc, Br, Lc + Cw + 0 * Inc, fg, bg, 0, 0, -1, 0, ""

```



```

FOR OldPos = 1 TO 4
    GOSUB DispNotSelected      ' display choices
NEXT OldPos
GOSUB ShowSelection          ' show choosen
OffCenter "[Left/Right] ARROW Keys for Choice = Then [ENTER] to Select", 25, 1, 80, 1, 7

BeamSelect$ = ""
WHILE BeamSelect$ <> CHR$(13)
    BeamSelect$ = INKEY$
    IF LEN(BeamSelect$) = 2 THEN BeamSelect$ = RIGHT$(BeamSelect$, 1)

    SELECT CASE BeamSelect$

        '---- Shift to the Left
        CASE CHR$(75)          ' Left Arrow key
            OldPos = BeamPos      ' Old Selected Position
            GOSUB DispNotSelected
            BeamPos = BeamPos - 1: IF BeamPos < 1 THEN BeamPos = 4
            GOSUB ShowSelection

        '---- Shift to the Right
        CASE CHR$(77)          ' Right Arrow key
            OldPos = BeamPos      ' Old Selected Position
            GOSUB DispNotSelected
            BeamPos = BeamPos + 1: IF BeamPos > 4 THEN BeamPos = 1
            GOSUB ShowSelection

        '---- On pressing enter key
        CASE CHR$(13)
            Z = SwitchBeam(BeamPos)      ' FUNC to switch the beam now!
            ScrnRestore TRow, BRow, Scr1(), Segment ' Restore original screen
            CurrentLocation = 1           ' Reset choice to grin alignment
            GOTO GetChoice                ' Show choices again

    END SELECT
WEND

CASE 3      '----- Select Auto Mode
    TRow = 1: BRow = 25      ' to be saved
    REDIM Scr1%(4000)
    DEF SEG = 0              ' Set default Segment to zero
    Segment = &HB800         ' For color graphics adapter
    ScreenSave TRow, BRow, Scr1(), Segment

    AutoWeldMode             ' enter the auto welding mode

    ScrnRestore TRow, BRow, Scr1(), Segment
    CurrentLocation = 1      ' Reset choice to 1st
    GOTO GetChoice

CASE 4      '----- Set System Date
    COLOR 1, 7: LOCATE 25, 1: Qprint0 STRING$(80, " "), -1 ' Clear 25th line
    LOCATE 25, 5
    INPUT ; "Enter New Date in mm-dd-yyyy Format ==> ", StrDate$
    ValidDate% = SetDate%(StrDate$)
    IF ValidDate = TRUE THEN Oops
    COLOR 7, 0
    GOTO GetChoice

CASE 5      '----- Set System Time
    COLOR 1, 7: LOCATE 25, 1: Qprint0 STRING$(80, " "), -1 ' Clear 25th line

```



```

LOCATE 25, 5
INPUT ; "Enter New Time in 24 hr. hh:mm:ss Format ==> ", StrTime$
ValidTime% = SetTime%(StrTime$)
IF ValidTime% = TRUE THEN Oops
COLOR 7, 0
GOTO GetChoice

CASE 6      '----- Quit to DOS
TRow = 1: BRow = 25      ' to be saved
REDIM Scr1%(4000)
DEF SEG = 0              ' Set default Segment to zero
Segment = &HB800          ' For color graphics adapter
ScreenSave TRow, BRow, Scr1(), Segment

CloseScreen              ' Sweep screen and display closing message

ScrRestore TRow, BRow, Scr1(), Segment
CurrentLocation = 1      ' Reset choice to 1st
GOTO GetChoice

'----- Draw Menu Screen
CASE ELSE
GOTO DrawScreen

END SELECT

ERASE Choices$, HelpText$, Scr1% ' Erase these DYNAMIC Arrays
SCREEN 0: COLOR 7, 0: CLS

END

'=====
'+++++++ Program End ++++++
'=====

'=====
'+++++++ Begin Subroutines ++++++
'=====

OpenScr1:
'-----
'This SUBroutine is a generic routine that opens a window on the screen.
'-----
REDIM Scr1%(4000)
DEF SEG = 0              ' Set default Segment to zero
Segment = &HB800          ' For color graphics adapter
ScreenSave TRow, BRow, Scr1(), Segment
MakeWindow TRow, LCol, BRow - 1, RCol, FColr, BColr, 0, 0, 0, 1, Label$
RETURN

'-----

ShowSelection:      ' show selected workstation for beam switching
MakeWindow Tr, Lc + (BeamPos - 1) * Inc, Br, Lc + Cw + (BeamPos - 1) * Inc, Sfg, Sbg, 0, 1, -1, 0, ""
LOCATE Tr + 1, 28 + (BeamPos - 1) * 7: Qprint0 Ws$(BeamPos), Sbg * 16 + Sfg
SOUND BeamPos * 100, .75
RETURN

'-----

DispNotSelected:    ' erase currently selected workstation for beam switch
MakeWindow Tr, Lc + (OldPos - 1) * Inc, Br, Lc + Cw + (OldPos - 1) * Inc, fg, bg, 0, 0, -1, 0, ""
LOCATE Tr + 1, 28 + (OldPos - 1) * 7: Qprint0 Ws$(OldPos), 1 * 16 + 11
RETURN

```



```

!-----
CarrierLoss:
    REDIM TempScr%(4000)
    Segment = &HB800
    ScreenSave 20, 25, TempScr(), Segment
    DispMsg "Loss of Carrier has been detected."
    Harddelay 5000                                ' hold msg a while
    ScrnRestore 20, 25, TempScr(), Segment          ' clear msg line
    CloseComm                                       ' close rs232 comm stuff
    RETURN      '????? or should it start over ????
' END      '??????????? IS THIS RIGHT ???
!-----

```

```

DEFSNG A-Z
SUB AutoWeldMode
DEFINT A-Z
!=====
' This sub is where the HOST performs the supervisory control of the Laser
' Welding process
!=====
Mode$ = "AUTO"                                ' set flag to AUTO mode
End$ = CHR$(0) + CHR$(79)                      ' code for END key (bail out if all inactive)

' - Set up Serial Port -
Port% = 1
WordLen% = 7
Parity% = 2      ' 0=none, 1=odd, 2=even
BaudRate% = 9600
Handshake% = 2    ' Handshake 0=none, 1=xon/xof, 2=xon/xoff and cts

' Comm port initialization call format:

OpenComm Port%, WordLen%, Parity%, BaudRate%, Handshake%

CarrierDetect 0      ' <-- Temporarily disable carrier detection
                        ' to allow data to be transmitted without
                        ' presence of CD signal

ON UEVENT GOSUB CarrierLoss
UEVENT ON
CarrierDetect 1
!*****
DispStat              ' draw the auto welding status screen

StartAutoWeld:

OkToQuit = 1          ' flag true indicates all stations INACTIVE
UpdateTime            ' show current time and date on status scr
    'read serial port for \Status data and enter or delete from que
ReQue                 ' call the sub that sets the que
UpdateStatDisp        ' show welder status
DO WHILE PortStatus = 1 ' if 1 then successful portstatus's found, 0=fail
    FOR W = 1 TO 4      ' get each WorkStation port's status
        IF PortStat(W + 1) = 1 THEN ' if port is alive try to connect and look for data
            ' 0 = not READY , 2=BUSY but READY
            Z = WsComm(W, "[WAIT FOR QUE ASSIGNMENT]" + CRLF$)
            ' if z=0 then no answer from ws!
        END IF
    NEXT W
    ReQue              ' find new position in QUE

```



```

' *- inform workstations of their que status (if>1) *-
NextWs = 0      ' clear Next Ws to be serviced
OkToQuit = 1    ' flag true indicates all stations INACTIVE

FOR W = 1 TO 4    ' find the ws that is later in que
  IF WStat(W).QuePos = 1 THEN
    NextWs = W    ' This is the next WS to be serviced
  ELSEIF WStat(W).QuePos <> 1 AND PortStat(W + 1) = 1 THEN ' If not next in Que and port is READY
    ' PortStat(X): 0=Not READY, 1=READY, 2=BUSY but READY
    Z = WsComm(W, "[WAIT] Que Assignment" + STR$(WStat(W).QuePos) + CRLF$)
    ' Send the Workstation's Que assignment
    IF Z = 0 THEN ReQue
  END IF
  IF INSTR(WStat(W).QueStat, "INACTIVE") = 0 THEN OkToQuit = 0 ' if INACTIVE not found its ok to check for q
uit
NEXT W

```

```

UpdateTime      ' show current time and date on status scr
UpdateStatDisp  ' show welder status

```

```

' *- get ready to weld with this workstation *-
' allow workstation to setup 701 laser welder
' then do the weld if setup fails abort

```

```

IF NextWs > 0 THEN      ' if a work station is in que weld
  QlineDisplayed = 0    ' quit msg disp'd has been erased
  LOCATE 25, 1: Qprint0 STRING$(80, " "), 113 ' Clear 25th line
  OffCenter "WS-" + RIGHT$(STR$(NextWs), 1) + " Setting Up Welder Parameters", 25, 1, 80, 1, 7

```

```

  IF SetupWs(NextWs) = 1 THEN ' if successful SetupWs lets weld

    LOCATE 25, 1: Qprint0 STRING$(80, " "), 113 ' Clear 25th line
    OffCenter "WS-" + RIGHT$(STR$(NextWs), 1) + " is Welding", 25, 1, 80, 1, 7

```

```

    Weldit (NextWs)      ' call the weld procedure now

    LOCATE 25, 1: Qprint0 STRING$(80, " "), 0   ' Clear 25th line

```

```

  END IF
END IF

```

```

!*****

```

```

UpdateTime      ' show current time and date on status scr
ReQue           ' call the sub that sets the que
UpdateStatDisp  ' show welder status

```

```

IF OkToQuit OR QlineDisplayed THEN GOTO StopCheck

```

```

LOOP
StopCheck:

```

```

  IF OkToQuit THEN      ' if quit is allowed
    IF QlineDisplayed = FALSE THEN ' if msg not disp'd
      LOCATE 25, 1: Qprint0 STRING$(80, " "), 113 ' Clear 25th line
      OffCenter "Press END to Return to Manual Mode", 25, 1, 80, 1, 7
      QlineDisplayed = 1
    END IF
    Key$ = INKEY$      ' look for END key
    IF Key$ = End$ THEN ' this is the only way out!
      Mode$ = "MANUAL"
      CloseComm      ' close communication port to prevent crashing!

```



```

        ClearDispVars ' clear variables used for display so re-entry will re fill them
        EXIT SUB      ' This is the exit on END!!!
    END IF
ELSE ' if can't quit (station welding)
    IF QlineDisplayed <> FALSE THEN ' if quit msg disp'd erase it
        LOCATE 25, 1: Qprint0 STRING$(80, " "), 0 ' Clear 25th line
        QlineDisplayed = 0
    END IF
END IF
GOTO StartAutoWeld

END SUB

DEFSNG A-Z
SUB BgDelay (MilliSecs!, MilliSecsToGo!, Status%, Dn%) STATIC
'-----
' BigDelay Procedure: Performs a background delay, returning control to the
' caller after each check so that the caller may be doing something while
' waiting.
' Modified to use milliseconds and also allow up to 10 delays to be running
' simultaneously (Dn%) indicates which timer is being used.
' Best Resolution is 54.945ms
'-----
' Input: an single precision representing the number of milli-seconds to delay
' Delay == Input: MilliSecs! = milli-seconds to wait
'       Output: MilliSecsToGo! = milli-seconds until timeout
'       Output: Status% = 0 - delay is complete; 1 - delay is continuing
'       Input: Dn% = delay number (unique to each sub or function last used
'                  was #6 (10-15-90)
' NOTE: once called do not reset Status
' Dn% assigned to these routines:
'       1 - SetupWs (2 times)
'       2 - Weldit
'       3 - WsComm
'       4 - GetSerialData
'       5 - RampLaser
'       6 - Weldit
'=====
DEFINT A-Z
DIM StartTime!(1 TO 10)
DIM EndingTime!(1 TO 10) ' Remember these values between calls

IF Status% = 0 THEN ' Entering for the first time this request
    StartTime!(Dn%) = Ticks(0) ' Mark the current time
    EndingTime!(Dn%) = Ticks(MilliSecs! / 10) ' Mark the ending time
    IF EndingTime!(Dn%) < StartTime!(Dn%) THEN
        EndingTime!(Dn%) = EndingTime!(Dn%) + TicksPerDay!
    END IF
END IF

CurrentTime! = Ticks(0)
IF CurrentTime! < StartTime! THEN
    CurrentTime! = CurrentTime! + TicksPerDay!
END IF

TicksLeft! = EndingTime!(Dn%) - CurrentTime! ' Calculate ticks left to go
MilliSecsToGo! = 10 * (TicksLeft! * 100) / TicksPerSec!

IF TicksLeft! > 0 THEN
    Status% = 1 ' Time has NOT run out
ELSE

```



```

        Status% = -999          ' Delay is complete (0 would automatically reset timer)
END IF

END SUB

SUB ClearDispVars
'--*-- Clear Display Variables --*-- so update stat display will work on re-entry

FOR I = 1 TO 4
    WStat(I).QueStat = ""
    WStat(I).ILock = ""          ' interlock status
    WStat(I).QueTime = DummyTime ' time rec'd from HP
    'WStat(I).QuePos = 5        ' position # in Q
NEXT I

    LStat(1).Power = ""          ' laser power status
    LStat(1).Remote = ""         ' laser in remote
    LStat(1).LReady = ""         ' laser ready (warmed up)
    LStat(1).PwrLimit = ""       ' power limit
    LStat(1).ETrip = ""          ' energy trip status

    IBoxStat(1).Remote = ""      ' box in remote
    IBoxStat(1).Air = ""         ' air pressure
    IBoxStat(1).Shutter = ""     ' shutter open / closed
    IBoxStat(1).SingShot = ""    ' single shot fired

    '"" BeamPos = 6              ' forces oldpos to be restored

END SUB

DEFSNG A-Z
SUB CloseScreen
'-----
' Close Screen Procedure: Draws the closing screen and displays a message.
'-----

DEFINT A-Z

'---- Draw Concentric Frames, with varying hues around outside border of screen.
fg = 4: bg = 0
Colr = bg * 16 + fg
FillScrn 1, 1, 25, 80, Colr, 176, 1          ' Color background with ""
FillScrn 2, 3, 24, 78, Colr, 177, 1          ' Color background with " "
FillScrn 3, 5, 23, 76, Colr, 178, 1          ' Color background with " "
FillScrn 4, 7, 22, 74, fg * 16 + fg, 177, 1  ' Color background with " "

SCREEN , , 1          ' Locate the cursor on Page 1
COLOR 15, 4
LOCATE 5, 19: PRINT "LUMONICS 701 LASER WELDER HOST COMPUTER"
LOCATE 6, 20: PRINT "PROGRAM WRITTEN BY B.EHRHART & J.GUSS"
LOCATE 7, 18: PRINT "ESPECIALLY FOR BELL LABS MODEL SHOP AT SSTC"
COLOR 14, 4
LOCATE 20, 27: PRINT "Press [ESC] to exit to DOS"
LOCATE 21, 37: PRINT "or"
LOCATE 22, 25: PRINT "[RETURN] to Return to Program"

SCREEN , , 0          ' Reset to Page 0

REDIM TempScr%(2000)
ScrnSave 1, 1, 25, 80, SEG TempScr%(1), 1

```


SweepScreen TempScr%(), 1

DO

```
a$ = INKEY$
SELECT CASE a$
CASE CHR$(27)
    CLS
    SYSTEM
CASE CHR$(13)      ' GOING TO RETURN TO PROGRAM
    ERASE TempScr%
    EXIT SUB
END SELECT
```

LOOP

END SUB

FUNCTION ConnectWs% (Ws)

```
'=====
' ConnectWs - function attempts up to 5 times to connect host to workstation
' (Ws) returns 1 for success, 0 for failure.
'=====
ConnectWs = 0      ' in case of failure to connect
DO      ' Loop up to 5 times to establish communications
    Transmit CHR$(5) + CHR$(13)      ' Send ^E CR to switch (a Wake up call)
    ClearInputBuffer      ' clear the data buffer
    Connect$ = "C HOST WS-" + LTRIM$(STR$(Ws))
    Transmit Connect$ + CHR$(13)      ' Make Connection thru Smart Switch to WS
    Harddelay (150)
    GetSerialData Return$      ' read data from Smart Switch
    IF INSTR(Return$, Connect$ + "1") > 0 THEN ' if connection was successful
        ConnectWs = 1      ' Successful connection
        Transmit CRLF$      ' ???
        EXIT FUNCTION      ' DO NOT Disconnect! Just Exit function
    END IF
    Tried = Tried + 1      ' inc failed attempt count
LOOP WHILE (Tried < 5)      ' try communications again
END FUNCTION
```

DEFSNG A-Z

```
'=====
' -- DispMsg -- Display one variable length msg
'      does not save or restore screen
'=====
```

SUB DispMsg (W\$)

DEFINT A-Z

```
Length = LEN(W$) + 1
IF Length > 0 THEN
    LastOne = 1
    TRow = 22 - LastOne: LCol = ((80 - Length) / 2)
    BRow = TRow + LastOne + 1: RCol = LCol + Length + 2: bg = 4
    fg = 15
    BRow = 25
    MakeWindow TRow, LCol, TRow + LastOne + 1, RCol, 14, 4, 0, 1, -1, 1, ""
    OffCenter W$, TRow + 1, LCol, RCol, fg, bg
```

END IF

END SUB

SUB DispStat


```
' fg% - The foreground color of normal menu entries. Allowable
' range is 0 to 15.
' bg% - The background color of normal menu entries. Allowable
' range is 0 to 7.
' hfg% - The foreground color of the highlighted menu entry. Allowable
' range is 0 to 15.
' hbg% - The background color of the highlighted menu entry. Allowable
' range is 0 to 7.
' qfg% - The foreground color of the Quick Access keys. Allowable
' range is 0 to 15.
' qbg% - The background color of the Quick Access keys. Allowable
' range is 0 to 7.
'/=====/'
```

```
hfg% = 14      ' highlighted foreground color
hbg% = 4       ' highlighted background color
qhc% = hbg% * 16 + hfg% ' highlighted text color for Qprint0
fg% = 15      ' foreground color
bg% = 4       ' background color
qnc% = bg% * 16 + fg%  ' normal text color for Qprint0
```

```
COLOR 4, 0      ' Color background of screen
ClrScr 1, ""    ' Use 176,177,or 178 for different hues
```

```
TRow = 1: BRow = 4: LCol = 10: RCol = 70
fg = 14: bg = 4
MakeWindow TRow, LCol, BRow, RCol, fg, bg, 0, 1, -1, 1, ""
OffCenter "BELL LABS SSTC LUMONICS JK 701 LASER STATUS", TRow + 1, 1, 80, 15, bg
OffCenter "AUTO MODE", TRow + 2, 1, 80, 15, bg
```

```
'--- Initialize Variables for Status Window
TRow = 5: BRow = 19: LCol = 7: RCol = 74
```

```
MakeWindow TRow, LCol, BRow, RCol, fg%, bg%, 0, 1, -1, 1, ""
```

```
LOCATE TRow + 1, LCol + 1: Qprint0 "                                WORKSTATION STATUS                                ", 64
LOCATE TRow + 2, LCol + 1: Qprint0 "                                WS-1      WS-2      WS-3      WS-4      ", qnc%
LOCATE TRow + 3, LCol + 1: Qprint0 "QUE STATUS =>                                ", qnc%
LOCATE TRow + 4, LCol + 1: Qprint0 "                                ", qnc%
LOCATE TRow + 5, LCol + 1: Qprint0 "DOOR INTERLOCK =>                                ", qnc%
LOCATE TRow + 6, LCol + 1: Qprint0 "                                ", qnc%
LOCATE TRow + 7, LCol + 1: Qprint0 "                                JK-701 LASER STATUS                                ", 64
LOCATE TRow + 8, LCol + 1: Qprint0 "AC POWER | REMOTE | READY | POWER LIMIT | ENERGY TRIP", qnc%
LOCATE TRow + 9, LCol + 1: Qprint0 "                                ", qnc%
LOCATE TRow + 10, LCol + 1: Qprint0 "                                ", qnc%
LOCATE TRow + 11, LCol + 1: Qprint0 "                                INTERFACE STATUS                                ", 64
LOCATE TRow + 12, LCol + 1: Qprint0 "REMOTE | AIR PRESS. | BEAM POS | SHUTTER | SINGLE SHOT", qnc%
LOCATE TRow + 13, LCol + 1: Qprint0 "                                ", qnc%
```

```
END SUB
```

```
FUNCTION DispWarning% (Warning$())
```

```
' Display Warning message (up to 5 lines) and return ASCII value for
' either [ESC] or [ENTER] when one of these is pressed
```

```
FOR I = 1 TO 5      ' SEARCH FOR LONGEST STRING
    IF Length < LEN(Warning$(I)) THEN Length = LEN(Warning$(I))
    IF LEN(Warning$(I)) > 0 THEN
        LastOne = I
```



```

        END IF
NEXT I
Length = Length + 1
TRow = 22 - LastOne: LCol = ((80 - Length) / 2)
BRow = TRow + LastOne + 1: RCol = LCol + Length + 2: bg = 4
fg = 15
BRow = 25
REDIM Scr1(4000)
DEF SEG = 0
Segment = &HB800
ScreenSave TRow, BRow, Scr1(), Segment
MakeWindow TRow, LCol, TRow + LastOne + 1, RCol, 14, 4, 0, 1, -1, 1, ""
FOR I = 1 TO LastOne
    OffCenter Warning$(I), TRow + I, LCol, RCol, fg, bg
    Warning$(I) = ""
NEXT I
LOCATE 25, 1: Qprint0 STRING$(79, " "), 116 ' clear bottom line
LOCATE 25, 20: Qprint0 "Press [ENTER] to Try Again OR [ESC] to Abort", 116
DO
    a$ = INKEY$
    IF LEN(a$) THEN      ' IF A KEY IS PRESSED
        IF ASC(a$) = 27 THEN
            ScrnRestore TRow, BRow, Scr1(), Segment
            DispWarning = 27
            EXIT FUNCTION
        ELSEIF ASC(a$) = 13 THEN
            ScrnRestore TRow, BRow, Scr1(), Segment
            DispWarning = 13
            EXIT FUNCTION
        END IF
    END IF
END IF
LOOP
END FUNCTION

DEFSNG A-Z
' ===== FILTER =====
'   Filters characters in an input string.
' =====
'
SUB Filter (InString$) STATIC

    ' Look for backspace characters and recode them to
    ' CHR$(29) (the LEFT cursor key):

    DO
        BackSpace = INSTR(InString$, CHR$(8))
        IF BackSpace THEN
            MID$(InString$, BackSpace) = CHR$(29)
        END IF
    LOOP WHILE BackSpace

    ' Look for line-feed characters and remove any found:
    DO
        DelChar10 = INSTR(InString$, CHR$(10))
        IF DelChar10 THEN
            InString$ = LEFT$(InString$, DelChar10 - 1) + MID$(InString$, DelChar10 + 1)
        END IF

        DelChar13 = INSTR(InString$, CHR$(13))
        IF DelChar13 THEN

```



```

        InString$ = LEFT$(InString$, DelChar13 - 1) + MID$(InString$, DelChar13 + 1)
    END IF

    DelChar5 = INSTR(InString$, CHR$(5))
    IF DelChar5 THEN
        InString$ = LEFT$(InString$, DelChar5 - 1) + MID$(InString$, DelChar5 + 1)
    END IF

    DelChar6 = INSTR(InString$, CHR$(6))
    IF DelChar6 THEN
        InString$ = LEFT$(InString$, DelChar6 - 1) + MID$(InString$, DelChar6 + 1)
    END IF

GOTO SKIP1

    LineFeed = INSTR(InString$, CHR$(10))
    IF LineFeed THEN
        InString$ = LEFT$(InString$, LineFeed - 1) + MID$(InString$, LineFeed + 1)
    END IF

SKIP1:
    LOOP WHILE DelChar5 OR DelChar6 OR DelChar10 OR DelChar13

END SUB

DEFINT A-Z
'=====
' *- GetSerialData *-
'=====
' Check the modem. If characters are waiting (DataWaiting is true)
' read them in.
'
SUB GetSerialData (ModemInput$)
    ModemInput$ = ""          ' clear the return variable

    TimeofLoop! = 10000      ' total of 10 seconds allowed for all data
    St = 0                   ' clear time for BgDelay (10 second loop)
    Dn = 4                   ' delay number unique to this sub !!!

    DO
        ModemInput$ = ModemInput$ + CHR$(ReadChar)  ' read in a char
        CALL BgDelay(TimeofLoop!, TimeLeft!, St, Dn) ' check for timeout
    LOOP UNTIL (St <= 0 OR DataWaiting = 0)
    Filter ModemInput$          ' Filter out line feeds and junk
END SUB

DEFSNG A-Z
SUB Harddelay (MilliSecs!) STATIC
DEFINT A-Z
'-----
' Delay Procedure: Performs a "hard" delay.
' input:  an integer representing the number of milli-seconds to delay
' delay == input: MilliSecs! = milli-seconds NOTE: Resolution to 54.95mS only!
'-----
StartTime! = Ticks(0)          ' mark the current time
EndingTime! = Ticks(MilliSecs! / 10) ' mark the ending time
IF EndingTime! < StartTime! THEN
    EndingTime! = EndingTime! + TicksPerDay!
END IF

DO
    CurrentTime! = Ticks(0)      ' mark the current time

```



```

    IF CurrentTime! < StartTime! THEN
        CurrentTime! = CurrentTime! + TicksPerDay!
    END IF
LOOP UNTIL CurrentTime! >= EndingTime!
'
END SUB

FUNCTION InitWelder%
' This function assures a known condition upon startup of the program
' It will initialize the 16 bit interface which will set the beam switch
' to position 4 and shut down all laser welder functions.
' More to follow
SHARED Warning$( )
IboxControlWord = BeamADD(4) + 0 ' init the control word
SetTimeout 1 ' set board timeout to 1 sec.

TryAgain:

KeyPress = 0 ' initialize the key to none
InitWelder% = 1 ' pass initialization

Writeio BOARDADD, 2, 0 ' initialize the board
ErrNum = Enter16(BOARDADD, IboxStatusWord) ' attempt to read interface box
' check for error with interface

SELECT CASE ErrNum
CASE 1601
    Warning$(1) = "WARNING!"
    Warning$(2) = "INTERFACE NOT FOUND"
    Warning$(3) = "AT ADDRESS " + STR$(BOARDADD)
    InitWelder% = 0 ' fail initialization
    KeyPress = DispWarning(Warning$( ))
CASE 1602
    Warning$(1) = "WARNING!"
    Warning$(2) = "16 Bit INTERFACE TIMEOUT"
    Warning$(3) = "Check Cable Between"
    Warning$(4) = "CPU and Interface Box"
    InitWelder% = 0 ' fail initialization
    KeyPress = DispWarning(Warning$( ))
CASE IS > 0
    Warning$(1) = "WARNING!"
    Warning$(2) = "16 Bit INTERFACE ERROR " + STR$(ErrNum)
    InitWelder% = 0 ' fail initialization
    KeyPress = DispWarning(Warning$( ))

CASE 0 ' If Interface Board is OK, Check Interface Box.

FOR I = 9 TO 11 ' Find Status Bit Read
    BitI = Bits(IboxStatusWord, I, I) ' BitI=1 If 2^I BIT IS SET
    SELECT CASE 2 ^ I ' Take a look at each bit
    CASE Remote ' 2 ^ 9 = 512 Bit
        IF BitI = 0 THEN
            Warning$(1) = "WARNING! REMOTE / LOCAL SWITCH MUST BE IN REMOTE"
            Warning$(2) = "BEFORE PROCEEDING"
            InitWelder% = 0 ' fail initialization
            KeyPress = DispWarning(Warning$( )) ' DISPLAY WARNING
        END IF
    CASE AirPress ' 2 ^ 10 = 1024 Bit
        IF BitI = 0 THEN
            Warning$(1) = "WARNING! LOW AIR PRESSURE"

```



```

Warning$(2) = "BEAM SWITCH CAN NOT BE SET AS DESIRED!"
InitWelder% = 0 ' fail initialization
KeyPress = DispWarning(Warning$()) ' DISPLAY WARNING

END IF
CASE LaserPower ' 2 ^ 11 = 2048 LASER AC Power ON/OFF Bit
IF BitI = 0 THEN
Warning$(1) = "WARNING! AC POWER TO LASER IS OFF!"
Warning$(2) = "TURN ON AC POWER (IN BASEMENT) BEFORE PROCEEDING"
InitWelder% = 0 ' fail initialization
KeyPress = DispWarning(Warning$()) ' DISPLAY WARNING
END IF
END SELECT
NEXT I
END SELECT
IF KeyPress = 13 THEN GOTO TryAgain
IF KeyPress = 27 THEN InitWelder% = 0
END FUNCTION

'=====
' *- OktoWeld *- Function checks laser and interface box to see if its ok
' to weld returns 0 if not OK, 1 if OK (no errors)
'=====
FUNCTION OktoWeld (Station)
HpMessage$ = ""
ErrNum = Enter16(BOARDADD, IboxStatusWord) ' read interface box
FOR I = 2 TO 11 ' Find Status Bit Read
BitI = Bits(IboxStatusWord, I, I) ' BitI=1 If 2^I BIT IS SET
SELECT CASE 2 ^ I ' Take a look at each bit
CASE LaserReady ' 2 ^ 1 = 2 Bit
IF BitI = 1 THEN HpMessage$ = "Laser not Ready,"
CASE LaserRemote ' 2 ^ 2 = 4 Bit
IF BitI = 1 THEN HpMessage$ = HpMessage$ + " Laser Not in REMOTE,"
CASE PowerLimit ' 2 ^ 3 = 8 Bit
IF BitI = 1 THEN HpMessage$ = HpMessage$ + " Laser Output Exceeds Power Limit,"
CASE EnergyTrip ' 2 ^ 4 = 16 Bit
IF BitI = 1 THEN HpMessage$ = HpMessage$ + " Laser Energy Trip,"
CASE 32 TO 256 ' USED FOR INTERLOCKS Bits 5-8
IF I - 4 = Station AND BitI = 0 THEN HpMessage$ = HpMessage$ + " Door Interlock is Open,"
CASE Remote ' 2 ^ 9 = 512 Bit
IF BitI = 0 THEN HpMessage$ = HpMessage$ + " Interface Box Not in REMOTE,"
CASE AirPress ' 2 ^ 10 = 1024 Bit
IF BitI = 0 THEN HpMessage$ = HpMessage$ + " Low Air Pressure to Beam Switch,"
CASE LaserPower ' 2 ^ 11 = 2048 LASER AC Power ON/OFF Bit
IF BitI = 0 THEN HpMessage$ = HpMessage$ + " Laser AC Power is OFF,"
END SELECT
NEXT I
IF HpMessage$ <> "" THEN ' if error messages exist send them!
Transmit LEFT$(">>> ERROR! " + HpMessage$ + "<<<", 75) + CRLF$
OktoWeld = 0 ' not ok to weld
PRINT HpMessage$, "work station="; Station
PRINT IboxStatusWord
ELSE ' otherwise say its ok to weld
OktoWeld = 1 ' it is ok to weld
END IF
END FUNCTION

DEFSNG A-Z
FUNCTION PortStatus%
DEFINT A-Z

```



```

PortStatus = 1          ' 1=success 0=fail
Transmit CHR$(10) + "+++" + CHR$(13)    ' Send Disconnect
Transmit CHR$(5) + CHR$(13)             ' Send ^E CR to switch
ClearInputBuffer                      ' clear the data buffer
Transmit "S" + CHR$(13)                 ' Send Status Request

Harddelay (150)

DO WHILE DataWaiting                ' read in the status array
    a$ = a$ + CHR$(ReadChar)         ' add to A$
LOOP                                ' end of reading in status array

X = INSTR(300, a$, "WS-1")           ' find position of WS-1
IF X THEN                           ' if something is returned
FOR I = 2 TO 5                       ' get only WorkStation port's status, put in array
    Ready$ = MID$(a$, X + 33 + ((I - 2) * 75), 3)    ' Device Ready Status
    PStatus$ = MID$(a$, X + 10 + ((I - 2) * 75), 4)  ' Port Status
    SELECT CASE Ready$                    ' return 0 for NOT READY, 1 for READY
        CASE "YES"
            IF PStatus$ = "FREE" THEN
                PortStat(I) = 1
            ELSEIF PStatus$ = "BUSY" THEN
                PortStat(I) = 2
            ELSE
                PortStat(I) = 999 ' Only if something is wrong!
            END IF
        CASE "NO "
            PortStat(I) = 0
        CASE ELSE                        ' Only if something is wrong!
            PortStat(I) = 999
            PortStatus = 0                ' 1=PortStatus OK, 0=Failed
        END SELECT
    NEXT I
ELSE
    PortStatus = 0                      ' if no a$ it failed
END IF
END FUNCTION

DEFSNG A-Z
FUNCTION Pround! (N!, PowerOfTen%)
'-----
' Pround FUNCTION: Returns the value of the Single precision argument (N!)
' rounded to a specified power of ten, as specified by the passed power of
' ten rounding value.
'-----
' Example of use: X! = Pround!(N!, PowerOfTen%)
' Examples:      Pround(136.789, -2) = 136.79
'               Pround(136.789, -1) = 136.8
'               Pround(136.789, 0)  = 137
'               Pround(136.789, 1)  = 140
'
' Parameters:    N!           Number to be rounded
'               PowerOfTen%   Power of 10 for rounding the number
' Variables:     pTen!        10 raised to the indicated power of 10

    pTen! = 10! ^ PowerOfTen%
    Pround! = FIX(N! / pTen! + .5) * pTen!
END FUNCTION

DEFINT A-Z
FUNCTION PWR% (Y)

```



```

'=====
'PWR returns the power of 2 that a number is raised to to = Y
'Example if: P=PWR(128), Then P=7
'=====

PWR% = LOG(Y) / LOG(2)
END FUNCTION

FUNCTION RampLaser% (Direction$)
'=====
'   -*- RampLaser -*-  RAMP LASER POWER UP or DOWN (return 1 for success)
'=====
StartRamp:

St = 0                ' clear time for BgDelay
Dn = 5                ' delay number unique to this sub !!!
TimeofLoop! = 15000   ' BgDelay 15 seconds

IF Direction$ = "UP" THEN                ' to Ramp UP
    IboxControlWord = IboxControlWord OR RampUp    ' add RampUp to data word out
ELSEIF Direction$ = "DOWN" THEN          ' to Ramp DOWN
    IboxControlWord = IboxControlWord AND -1 - RampUp ' RampUp if set
END IF
DO                                        ' loop till laser ready or timeout
    Z = Output16(BOARDADD, IboxControlWord)        ' select the Ramp now!
    Z = Enter16(BOARDADD, IboxStatusWord)          ' read status of ibox
    SELECT CASE Bits(IboxStatusWord, PWR(LaserRemote), PWR(LaserReady)) ' bit 1=LaserReady bit 2=LaserRemote
        CASE 0 ' all is ok
            RLflag = 1
        CASE 2
            RLflag = 0
            Warning$(1) = "WARNING! LASER IS NOT IN REMOTE!"
            Warning$(2) = "USE LASER CONTROL PENDANT"
            Warning$(3) = "AND SET LASER TO REMOTE!"
        CASE 1
            IF Direction$ = "UP" THEN ' failed power up
                RLflag = 0
                Warning$(1) = "WARNING! RAMP UP FAILED!"
            ELSEIF Direction$ = "DOWN" THEN RLflag = 1 ' successful power up
            END IF
        CASE 3
            IF Direction$ = "DOWN" THEN ' failed power down
                RLflag = 0
                Warning$(1) = "WARNING! RAMP DOWN FAILED!"
            ELSEIF Direction$ = "UP" THEN RLflag = 1 ' successful power up
            END IF
    END SELECT
    CALL BgDelay(TimeofLoop!, TimeLeft!, St, Dn) ' check for timeout
    IF St > 0 AND Mode$ = "MANUAL" THEN
        IF Md = 0 THEN 'if no message displayed
            REDIM TempScr%(4000)
            Segment = &HB800
            ScreenSave 20, 25, TempScr%(), Segment
            Md = 1 ' message display flag set
        END IF
        DispMsg "Waiting for Laser Ready" + STR$(Pround(TimeLeft! \ 1000, -1)) + " Seconds"
    END IF
LOOP UNTIL St <= 0 OR RLflag = 1 ' keep trying until timeout or laser ready
IF Md THEN ' if message displayed restore screen!
    ScrnRestore 20, 25, TempScr%(), Segment

```



```

END IF

IF Mode$ = "MANUAL" AND RLflag = 0 THEN
    KeyPress = DispWarning(Warning$())
    IF KeyPress = 13 THEN GOTO StartRamp
    IF KeyPress = 27 THEN RampLaser = RLflag: EXIT FUNCTION
ELSEIF Mode$ = "AUTO" AND RLflag = 0 THEN
    Transmit ">>> FAIL RAMP-" + Direction$ + CRLF$
END IF
RampLaser = RLflag
END FUNCTION

'===== ReQue =====
' Finds the current Queue order based on the variable QueTime!() and
' stores the resultant Queue order in the QueStat() array
' 1-4=Que position (1 is ready to weld) 5=Canceled or Inactive
'=====
SUB ReQue

DIM Temp(4) AS DOUBLE ' Temporary storage for the incremental (low to high) order
                        ' when sorted
FOR I = 1 TO 4 ' assign times to Temp() and clear QueStat()
    Temp(I) = WStat(I).QueTime
    IF WStat(I).QueTime > 2.14E+11 THEN WStat(I).QuePos = 5 ELSE WStat(I).QuePos = 99
    ' time > 2.14E+11 is ~ year 2070
NEXT I

'=====
' The following is copied from the QB SortDemo program it is the sub
' named ExchangeSort and is modified slightly here
'=====
FOR I = 1 TO 4
    SmallestRow = I
    FOR J = I + 1 TO 4
        IF Temp(J) < Temp(SmallestRow) THEN SmallestRow = J
    NEXT J
    IF SmallestRow > I THEN SWAP Temp(I), Temp(SmallestRow)
NEXT I

'===== End ExchangeSort =====

FOR I = 1 TO 4 ' find each QuePos by finding the matching QueTime and Temp
    IF WStat(I).QuePos <> 5 THEN
        FOR J = 1 TO 4
            IF WStat(I).QueTime = Temp(J) THEN WStat(I).QuePos = J: J = 4
            SELECT CASE WStat(I).QuePos ' fill in the status$
                CASE 1
                    WStat(I).QueStat = " WELDING "
                CASE 2 TO 4
                    WStat(I).QueStat = " " + STR$(WStat(I).QuePos) + " IN QUE "
            END SELECT
        NEXT J
    ELSE
        WStat(I).QueStat = " INACTIVE "
    END IF
NEXT I

'FOR I = 1 TO 4
' PRINT WStat(I).QueTime, WStat(I).QuePos, WStat(I).QueStat, Temp(I)
'NEXT I

```


END SUB

FUNCTION SetupWs% (Ws)

```
'=====
'  *- SetupWs *- Establish Workstation Communication
'
'      Ws=WorkStation number, WsMessage$=Message to be sent if
'      CANCEL is not received!
'
'      Will make 5 attempts to communicate. Time limit for data to
'      be received is determined in Get Serial Data routine.
'
'      Func returns 1 if all is ok 0 if CANCEL was received.
'=====
```

```
Tried = 0                ' Clear # of times connection was tried
SetupWs = 0              ' setup to return failure flag unless success
```

```
Transmit CHR$(5) + CHR$(13)    ' Send ^E CR to switch (a Wake up call)
ClearInputBuffer              ' clear the data buffer
```

```
IF ConnectWs(Ws) = 0 THEN GOTO FailExit    ' make connection to Ws, if not successful SetupWs will return a 0
```

```
IF Ramplaser("UP") = 0 THEN GOTO FailExit    ' added because!
IF SwitchBeam(Ws) = 0 THEN GOTO FailExit    ' added because!
```

```
DO                                ' Loop up to 5 times to establish communications
    Transmit CRLF$ + "[SETUP]" + CRLF$    ' attempt to get HP's attention
    '  *- wait for data from work station *-
    St = 0                            ' clear time for BgDelay
    Dn = 1                            ' unique delay number for this function
    TimeofLoop! = 2000                ' BgDelay 2 seconds
    DO                                ' loop till data is xmitted or time is up
        CALL BgDelay(TimeofLoop!, TimeLeft!, St, Dn)    ' check for timeout
    LOOP UNTIL (St <= 0 OR DataWaiting)    ' 2 second limit
    '  *- if data is returned from work station *-
    IF DataWaiting THEN                ' if data is found
        GetSerialData Return$          ' Read xmitted data
        ' Valid responses are "[SET]" or "[CANCEL]"
```

```
    IF LEFT$(Return$, 5) = "[SET]" THEN
        Transmit CHR$(10) + "+++" + CHR$(13)    ' Send Disconnect
        ClearInputBuffer                    ' just clear the buffer???

        Harddelay 1000                    ' wait 1 second for hp to make its connection
        St = 0                            ' clear time for BgDelay
        Dn = 1                            ' unique delay number for this function
        TimeofLoop! = 60000                ' wait up to 60 seconds for port to respond again
```

TryPSagain:

```
DO                                ' loop until port becomes ready or timeout
    Z = PortStatus                    ' if 1 then successful portstatus's found, 0=fail
    CALL BgDelay(TimeofLoop!, TimeLeft!, St, Dn)    ' check if time is left
    LOOP UNTIL (St <= 0 OR PortStat(Ws + 1) = 1)    ' timeout or port is READY
    ' 0=Not READY, 1=READY, 2=BUSY but READY
    IF St <= 0 THEN                    ' if timeout occurs
        GOTO FailExit                ' then Exit function
    ELSEIF PortStat(Ws + 1) = 1 THEN    ' if port becomes ready
        IF ConnectWs(Ws) = 1 THEN      ' make connection to Ws, if successful
            SetupWs = 1                ' Successful connection
            EXIT FUNCTION              ' DO NOT Disconnect! Just Exit function
        ELSE                            ' if connection fails
            GOTO TryPSagain            ' WAS GOTO FailExit (BUT! for some reason PortStat sometimes
```

returned bogus "READY" status)


```

        END IF
        ' No other conditions could get here! Either times out or port is ready
    END IF
    ELSEIF LEFT$(Return$, 8) = "[CANCEL]" THEN
        GOTO FailSetupWs          ' Disconnect and Exit function
    END IF
    ' neither SET or CANCEL rec'd
    END IF
    ' if no data is found (upomn switch connection)
    ' only gets here if no data shows up in the 2 second wait
    Tried = Tried + 1          ' inc failed count
    LOOP WHILE (Tried < 5)      ' try communications again

FailSetupWs:                ' if timeout or cancel
Transmit "[QUIT]" + CRLF$
Transmit CHR$(10) + "+++" + CHR$(13)    ' Send Disconnect

FailExit:                    '
X = RampLaser("DOWN")      ' added because!
ClearInputBuffer            ' just clear the buffer???
WStat(Ws).QueTime = 9E+11    ' Clear the Que This is a long time away

END FUNCTION

SUB SweepScreen (Scr%(), El%) STATIC
'-----
' Sweep Screen Procedure: Clears screen by
'-----
DEFINT A-Z
    E = -1 + El

    FOR Col = 40 TO 80 STEP 8
        Slope! = (Col - 40) / 12
        FOR row = 1 TO 12
            SCol = INT((13 - row) * Slope!) + 40
            MPRestore row, 40, row, SCol, 80, Scr((row - 1) * 80 + 40 + E)
        NEXT
    NEXT

    FOR row = 2 TO 12 STEP 1
        Slope! = 40 / (13 - row)
        FOR R = row TO 12
            SCol = INT((13 - R) * Slope!) + 40
            IF SCol > 80 THEN SCol = 80
            MPRestore 1, 40, R, SCol, 80, Scr(40 + E)'Scr((R - 1) * 80 + 40+E)
        NEXT
    NEXT

    FOR row = 13 TO 25 STEP 1
        Slope! = 40 / (row - 12)
        FOR R = 13 TO row
            SCol = INT((R - 12) * Slope!) + 40
            IF SCol > 80 THEN SCol = 80
            MPRestore R, SCol, R, 80, 80, Scr(((R - 1) * 80) + SCol + E)
        NEXT
    NEXT

    FOR Col = 73 TO 1 STEP -8
        Slope! = (Col - 39) / 13
        FOR row = 13 TO 25
            SCol = INT((row - 12) * Slope!) + 39
            MPRestore row, SCol, row, 80, 80, Scr((row - 1) * 80 + SCol + E)

```



```

NEXT
NEXT

FOR row = 24 TO 13 STEP -1
    Slope! = 39 / (row - 12)
    FOR R = 13 TO row
        SCol = 39 - INT((R - 12) * Slope!)
        IF SCol < 1 THEN SCol = 1
        MPRestore R, SCol, R, 39, 80, Scr((R - 1) * 80 + SCol + E)
    NEXT
NEXT

FOR row = 12 TO 1 STEP -1
    Slope! = 39 / (row - 13)
    FOR R = 12 TO row STEP -1
        SCol = 39 - INT((R - 13) * Slope!)
        IF SCol < 1 THEN SCol = 1
        MPRestore R, 1, R, SCol, 80, Scr((R - 1) * 80 + 1 + E)
    NEXT
NEXT

FOR Col = 8 TO 40 STEP 8
    Slope! = (39 - Col) / 12
    FOR row = 1 TO 12
        SCol = 39 - INT((13 - row) * Slope!)
        MPRestore row, 1, row, SCol, 80, Scr((row - 1) * 80 + 1 + E)
    NEXT
NEXT

END SUB

DEFSNG A-Z
'=====
' *- SwitchBeam *- Function returns 0 if fail, 1 if succesful (no errors)
'=====
FUNCTION SwitchBeam% (DesiredWorkStation%)

DEFINT A-Z
SHARED Warning$()
SHARED Mode$

IF Mode$ = "MANUAL" THEN
ReadStatus: Z = Enter16(BOARDADD, IboxStatusWord)
    KeyPress = 0          ' used to determine status if key is pressed after warning msg.
    IBox = Bits(IboxStatusWord, 10, 9)    ' look at bits 9 and 10 only
        ' return 0 if neither is set 1 if bit 9 (Remote) is
        ' set, 2 if bit 10 (AirPress) and 3 if both are set

    SELECT CASE IBox
    CASE 0                ' IF LOW AIR AND NOT REMOTE
        SwitchBeam = 0    ' Fail to set beam switch
        Warning$(1) = "WARNING! INTERFACE BOX MAY BE TURNED OFF"
        Warning$(2) = "or REMOTE / LOCAL SWITCH IS IN LOCAL POSITION"
        Warning$(3) = "and AIR PRESSURE IS TOO LOW!"
        Warning$(4) = "CORRECT THE ABOVE CONDITION(S) TO PROCEED"
        KeyPress = DispWarning(Warning$())    ' DISPLAY WARNING
    CASE 1                ' IF LOW AIR PRESSURE
        SwitchBeam = 0    ' Fail to set beam switch
        Warning$(1) = "WARNING! LOW AIR PRESSURE"
        Warning$(2) = "BEAM SWITCH CAN NOT BE SET AS DESIRED!"
        KeyPress = DispWarning(Warning$())    ' DISPLAY WARNING

```



```

CASE 2          ' IF BOX NOT IN REMOTE
    SwitchBeam = 0      ' Fail to set beam switch
    Warning$(1) = "WARNING! REMOTE / LOCAL SWITCH MUST BE IN REMOTE"
    Warning$(2) = "BEFORE PROCEEDING"
    Warning$(3) = "PRESS RETURN TO CONTINUE"
    KeyPress = DispWarning(Warning$())      ' DISPLAY WARNING
CASE 3          ' AIR PRESSURE OK AND IN REMOTE so... SELECT BEAM!
    GOTO Switchit      ' and exit
END SELECT
IF KeyPress = 13 THEN GOTO ReadStatus      ' try it again
IF KeyPress = 27 THEN EXIT FUNCTION      ' abort
ELSE      ' if mode = AUTO
    IF OktoWeld(DesiredWorkStation) = 1 THEN
        GOTO Switchit      ' and exit
    ELSE      ' if not ok to weld then switchbeam will fail
        SwitchBeam = 0      ' failed (auto mode)
        EXIT FUNCTION      ' abort with fail flag
    END IF
END IF
Switchit:      ' AIR PRESSURE OK AND IN REMOTE so... SELECT BEAM!
    SwitchBeam = 1      ' Success setting beam switch
    IboxControlWord = IboxControlWord AND -385      ' clear any beam selected
    IboxControlWord = IboxControlWord OR BeamADD(DesiredWorkStation)      ' add selected beam to data word out
    Z = Output16(BOARDADD, (IboxControlWord))      ' select the beam now!
    BeamPos = DesiredWorkStation
END FUNCTION

```

DEFSNG A-Z

FUNCTION Ticks! (HundSecs!) STATIC

```

'-----
' Function Ticks! -- returns the number of clock ticks since midnight
'
' Allows the caller to 'mark' the time, with an optional offset
' ahead in time. Very useful in timing routines.
'
' Usage:
'   HundSecs = 100
'   Mark! = Ticks (HundSecs)      ' # of clock ticks that will have occurred
'                                   ' since midnite (in 1 sec. from now)
'
' Input: HundSecs = hundredths of seconds from now
' Output: TimeMark! = time marker calculated (clock ticks since midnight)
'-----

```

```

'---- This DIM instruction assigns InputRegs and OutputRegs to RegType, in
' this procedure. It could have been placed in the main program
' immediately after defining the register data type.

```

DIM InputRegs AS RegType, OutputRegs AS RegType

```

'---- The AX register is made up of AH on the left and AL on the right.
' The AH section is referred to by taking the value to be sent to
' that register and multiplying it by 256 in decimal or 100 in hex.
' For example, a value of 2 or &H02 sent to an AH register will be
' sent as (2 * 256) or &H200. Using this system, an AH and an AL
' value can be sent simultaneously by adding the AL value to the
' AH value. AH = 2 and AL = 3 becomes &H200 + &H3 or (256 * 2) + 3
'-----

```

```

    InputRegs.ax = &H0      ' Set AH to 0 to get time of day (and AL, which is
                            ' the midnight flag, will be reset to 0 if clock
                            ' was read or written, via AH = 0 or 1, within the
                            ' current 24 hour period. Otherwise, AL > 0)

```


CALL INTERRUPT(&H1A, InputRegs, OutputRegs) ' BIOS Call &H1A, service 0

'---- Return CX:DX, will equal the clock count (low-order word in DX)

cx! = OutputRegs.cx: dx! = OutputRegs.dx

ClkTicks = (cx! * 2 ^ 16 + dx!) + (HundSecs! / 100 * TicksPerSec!)

'---- Adjust for turnover, passed midnite

IF ClkTicks > TicksPerDay! THEN ClkTicks = ClkTicks - TicksPerDay!

Ticks! = FIX(ClkTicks)

END FUNCTION

DEFINT A-Z

SUB UpdateStatDisp

'=====

' Display Laser, Interlock and Workstation Status

'=====

DIM OldWSstat(4) AS INTEGER

'---- Initialize Variables for Status Window

TRow = 5: BRow = 19: LCol = 7: RCol = 74

hfg = 14 ' highlighted foreground color

hbg = 4 ' highlighted background color

qhc = hbg * 16 + hfg ' highlighted text color for Qprint0

fg = 15 ' foreground color

bg = 4 ' background color

qnc = bg * 16 + fg ' normal text color for Qprint0

' Line#

1	WORKSTATION STATUS				
2	WS-1	WS-2	WS-3	WS-4	
3	QUE STATUS =>	WELDING	2 IN QUE	3 IN QUE	INACTIVE
4					
5	DOOR INTERLOCK =>	CLOSED	CLOSED	OPEN	OPEN
6					
7	JK-701 LASER STATUS				
8	AC POWER	REMOTE	READY	POWER LIMIT	ENERGY TRIP
9	ON	YES	YES	OK	OK
10					
11	INTERFACE STATUS				
12	REMOTE	AIR PRESS.	BEAM POS	SHUTTER	SINGLE SHOT
13	YES	OK	WS-1	CLOSED	NO

Z = Enter16(BOARDADD, IboxStatus) ' Read Status of Interlocks

FOR I = 1 TO 11 ' Find Status Bit Read

BitI = Bits(IboxStatus, I, 1) ' BitI=1 If 2^I BIT IS SET

SELECT CASE 2 ^ I ' Take a look at each bit

CASE LaserReady ' 2 ^ 1 = 2 Bit

IF BitI = 1 THEN ' if bit is set LOW

IF LStat(1).LReady <> " NO " THEN LStat(1).LReady = " NO " " : LOCATE TRow + 9, LCol + 28: Qprint0 LStat(1).LReady, qhc

ELSE ' if bit is set HI

IF LStat(1).LReady <> " YES " THEN LStat(1).LReady = " YES " " : LOCATE TRow + 9, LCol + 28: Qprint0 LStat(1).LReady, qhc

END IF ' end test for this bit

CASE LaserRemote ' 2 ^ 2 = 4 Bit

IF BitI = 1 THEN ' if bit is set LOW

IF LStat(1).Remote <> " NO " THEN LStat(1).Remote = " NO " " : LOCATE TRow +


```

9, LCol + 15: Qprint0 LStat(1).Remote, qhc
      ELSE                                ' if bit is set HI
      IF LStat(1).Remote <> "    YES    " THEN LStat(1).Remote = "    YES    " ": LOCATE Trow +
9, LCol + 15: Qprint0 LStat(1).Remote, qhc
      END IF                              ' end test for this bit
CASE PowerLimit                          ' 2 ^ 3 = 8 Bit
  IF BitI = 1 THEN                        ' if bit is set LOW
    IF LStat(1).PwrLimit <> "    FAIL   " THEN LStat(1).PwrLimit = "    FAIL   " ": LOCATE TR
ow + 9, LCol + 41: Qprint0 LStat(1).PwrLimit, qhc
    ELSE                                  ' if bit is set HI
    IF LStat(1).PwrLimit <> "    OK    " THEN LStat(1).PwrLimit = "    OK    " ": LOCATE TR
ow + 9, LCol + 41: Qprint0 LStat(1).PwrLimit, qhc
    END IF                              ' end test for this bit
CASE EnergyTrip                          ' 2 ^ 4 = 16 Bit
  IF BitI = 1 THEN                        ' if bit is set LOW
    IF LStat(1).ETrip <> "    FAIL   " THEN LStat(1).ETrip = "    FAIL   " ": LOCATE Trow + 9
, LCol + 54: Qprint0 LStat(1).ETrip, qhc
    ELSE                                  ' if bit is set HI
    IF LStat(1).ETrip <> "    OK    " THEN LStat(1).ETrip = "    OK    " ": LOCATE Trow + 9
, LCol + 54: Qprint0 LStat(1).ETrip, qhc
    END IF                              ' end test for this bit
CASE 32 TO 256                          ' USED FOR INTERLOCKS Bits 5-8
  W = I - 4                              ' find WorkStation Number
  IF BitI = 1 THEN                        ' if bit is set LOW
    IF WStat(W).ILock <> "    CLOSED " THEN WStat(W).ILock = "    CLOSED " ": LOCATE Trow + 5,
LCol + 20 + (W - 1) * 12: Qprint0 WStat(W).ILock, qhc
    ELSE
    IF WStat(W).ILock <> "    OPEN  " THEN WStat(W).ILock = "    OPEN  " ": LOCATE Trow + 5,
LCol + 20 + (W - 1) * 12: Qprint0 WStat(W).ILock, qhc
    END IF
CASE Remote                              ' 2 ^ 9 = 512 Bit
  IF BitI = 1 THEN                        ' if bit is set LOW
    IF IBoxStat(1).Remote <> "    YES   " THEN IBoxStat(1).Remote = "    YES   " ": LOCATE
Trow + 13, LCol + 1: Qprint0 IBoxStat(1).Remote, qhc
    ELSE                                  ' if bit is set HI
    IF IBoxStat(1).Remote <> "    NO    " THEN IBoxStat(1).Remote = "    NO    " ": LOCATE
Trow + 13, LCol + 1: Qprint0 IBoxStat(1).Remote, qhc
    END IF                              ' end test for this bit
CASE AirPress                            ' 2 ^ 10 = 1024 Bit
  IF BitI = 1 THEN                        ' if bit is set LOW
    IF IBoxStat(1).Air <> "    OK    " THEN IBoxStat(1).Air = "    OK    " ": LOCATE Trow
+ 13, LCol + 15: Qprint0 IBoxStat(1).Air, qhc
    ELSE                                  ' if bit is set HI
    IF IBoxStat(1).Air <> "    LOW   " THEN IBoxStat(1).Air = "    LOW   " ": LOCATE Trow
+ 13, LCol + 15: Qprint0 IBoxStat(1).Air, qhc
    END IF                              ' end test for this bit
CASE LaserPower                          ' 2 ^ 11 = 2048 LASER AC Power ON/OFF Bit
  IF BitI = 1 THEN                        ' if bit is set LOW
    IF LStat(1).Power <> "    ON    " THEN LStat(1).Power = "    ON    " ": LOCATE Trow + 9
, LCol + 1: Qprint0 LStat(1).Power, qhc
    ELSE                                  ' if bit is set HI
    IF LStat(1).Power <> "    OFF   " THEN LStat(1).Power = "    OFF   " ": LOCATE Trow + 9
, LCol + 1: Qprint0 LStat(1).Power, qhc
    END IF                              ' end test for this bit
END SELECT
NEXT I
LOCATE Trow + 13, LCol + 28: Qprint0 "    WS-" + RIGHT$(STR$(BeamPos), 1), qhc

' SHUTTER LINE->' LOCATE Trow + 13, LCol + 41: QPRINT0 "    WS-" + RIGHT$(STR$(BeamPos), 1), qhc
' SING SHOT LINE->' LOCATE Trow + 13, LCol + 54: QPRINT0 "    WS-" + RIGHT$(STR$(BeamPos), 1), qhc

```



```

FOR W = 1 TO 4
    IF OldWSstat(W) <> WStat(W).QuePos THEN
        OldWSstat(W) = WStat(W).QuePos
        LOCATE TRow + 3, LCol + 20 + (W - 1) * 12: Qprint0 WStat(W).QueStat, qhc
    END IF
NEXT W
END SUB

```

SUB UpdateTime

```

'+++ Display Time & Date on the Status screen +++
hr = VAL(LEFT$(TIME$, 2))
'---- Time is displayed in 12 hr format with Am/Pm.
IF hr = 0 THEN                                ' It is between midnight and 1 am
    hr = 12
    AMorPM$ = " am"
ELSEIF hr > 12 THEN                            ' After 1:00 pm
    hr = hr - 12                              ' Conversion to 12 hr. format.
    AMorPM$ = " pm"
ELSEIF hr = 12 THEN                            ' Between noon and 1:00 pm
    AMorPM$ = " pm"
ELSE                                            ' Anything else is between 1:00 am
    AMorPM$ = " am"                          ' and noon.
END IF

'---- Adjust column that time is printed on.
SELECT CASE hr
    CASE 1 TO 9
        timecol = 58                          ' IF Hr is 1 thru 9.
        LOCATE 3, 57                          ' Clear leading space when hr
        Qprint0 " ", 68                      ' switches from 12 to 1.
    CASE ELSE
        timecol = 57                          ' IF Hr is 10,11,12
END SELECT

'---- Display Time
CurTime$ = STR$(hr) + RIGHT$(TIME$, 6) + AMorPM$
LOCATE 3, timecol
Qprint0 CurTime$ + " ", 74

'---- Build & Display date string
Month$ = MonthName$(VAL(LEFT$(DATE$, 2)))
CurrentDate$ = Month$ + " " + MID$(DATE$, 4, 2) + " " + MID$(DATE$, 7, 4)
LOCATE 3, 11
Qprint0 " " + CurrentDate$ + " ", 74

```

END SUB

DEFSNG A-Z

SUB Weldit (Station%)

```

'=====
'  *- WELDIT *-          DO THE WELDING
'=====

```

DEFINT A-Z

```

FailedCount = 0                                ' count of number of fails
St = 0                                          ' Set to 0 to start background timing
Dn = 2                                          ' delay number unique to this sub!
TimeToWait! = 2 * 60 * 1000!                 ' 2 minutes (*60*1000 converts to milli-secs)

```

UpdateStatDisp ' show status now


```
CALL BgDelay(TimeToWait!, TimeToGo!, St, Dn) ' Start background timing
```

```
WeldItStart:
```

```
Transmit "[READY]" + CRLF$ ' Send ready to weld to the HP
```

```
DO
```

```
    St2 = 0 ' Set to 0 to start background timing
```

```
    Dn2 = 6 ' delay number unique to this sub!
```

```
    TimeToWait2! = 45000 ' 45 seconds for data to come back
```

```
DO
```

```
    CALL BgDelay(TimeToWait2!, TimeToGo2!, St2, Dn2) ' see if time is left till data comes in
```

```
    LOOP WHILE DataWaiting = 0 AND St2 > 0
```

```
    IF St2 <= 0 THEN GOTO QuitWeld
```

```
    IF DataWaiting THEN
```

```
        GetSerialData Return$ ' Read xmitted data
```

```
        SELECT CASE LEFT$(Return$, 5) ' look at 5 left most chars returned
```

```
            CASE "[YES]"
```

```
                ' IF RampLaser("UP") = 0 THEN GOTO WeldItStart ' Fire up laser, if fail try again
```

```
                ' IF SwitchBeam(Station) = 0 THEN GOTO WeldItStart ' Switch beam, if fail try again
```

```
                    Transmit "[BEGIN]" + CRLF$ ' tell hp to start
```

```
                    GOTO WeldLoop ' start welding
```

```
            CASE "[CANC"EL] ' if sent in response to ready
```

```
                HpMessage$ = "[QUIT]"
```

```
                GOTO ExitWeldit ' prepare to leave weldit
```

```
        END SELECT
```

```
    ELSE
```

```
        FailedCount = FailedCount + 1 ' inc if no data waiting
```

```
    END IF
```

```
    CALL BgDelay(TimeToWait!, TimeToGo!, St, Dn)' check background timing
```

```
    LOOP WHILE St > 0 AND FailedCount < 5
```

```
    GOTO QuitWeld ' gets here if time out or failedcount too high
```

```
WeldLoop:
```

```
DO ' loop till [FIRE] is not returned
```

```
Harddelay (150)
```

```
IF DataWaiting THEN
```

```
    GetSerialData Return$ ' Read xmitted data
```

```
    SELECT CASE LEFT$(Return$, 6) ' look at 6 left most chars returned
```

```
        CASE "[FIRE]" ' fire single shot
```

```
            IF OktoWeld(Station) = 1 THEN ' function checks status of welder and interfacebox
```

```
                Z = Output16(BOARDADD, (IboxControlWord + SingleShot))
```

```
                Harddelay 200 ' wait 200 ms
```

```
                Z = Output16(BOARDADD, IboxControlWord)
```

```
                Transmit "[OK]" + CRLF$
```

```
            END IF
```

```
        CASE "[WAIT]" 'Communication just to keep smart switch happy!
```

```
            Transmit "[OK]" + CRLF$
```

```
        CASE "[DONE]" ' ready to stop welding
```

```
            HpMessage$ = "[O&O]" ' over and out - sign off
```

```
            GOTO ExitWeldit ' prepare to leave weldit
```

```
        CASE "[CANCE"EL] ' if sent in response to ready
```

```
            HpMessage$ = "[QUIT]"
```

```
            GOTO ExitWeldit ' prepare to leave weldit
```

```
        CASE ELSE ' Bogus data rec'd!
```

```
            FailedCount = FailedCount + 1 ' inc failcount
```

```
        END SELECT
```

```
    END IF
```

```
    CALL BgDelay(TimeToWait!, TimeToGo!, St, Dn) ' check background timing
```

```
    LOOP WHILE St > 0 AND FailedCount < 5
```

```
QuitWeld:
```



```

FailedCount = FailedCount + 1      ' falls in here if weld time is used up
CALL BgDelay(TimeToWait!, TimeToGo!, St, Dn)      ' check background timing
IF FailedCount < 5 AND St > 0 THEN      ' if fail count is ok and time is left
    Transmit "[RETRY]" + CRLF$      ' tell hp lets try again
    GOTO WeldItStart      ' retry
ELSE      ' failed for timeout or too many errors
    HpMessage$ = "[QUIT] Aborting Weld Cycle"
    IF St <= 0 THEN HpMessage$ = HpMessage$ + " 5 minute weld cycle is up!"
    IF FailedCount >= 5 THEN HpMessage$ = HpMessage$ + " too many transmission errors!"
    GOTO ExitWeldit
END IF

ExitWeldit:      ' for timeout or cancel or normal exits
X = RampLaser("DOWN")      ' turn off laser !!!
' should ramp down if que is empty and be done in autoweldmode not here!!!

WStat(Station).QueTime = DummyTime      ' reset time for que
Transmit HpMessage$ + CRLF$      ' send message to hp
Transmit CHR$(10) + "+++" + CHR$(13)      ' Send Disconnect
ClearInputBuffer      ' just clear the buffer???

END SUB

FUNCTION WsComm% (Ws, WsMessage$)
'=====
'  -- WsComm -- Workstation Communication
'      Ws=WorkStation number, WsMessage$=Message to be sent if
'      CANCEL is not received!
'      Will make 5 attempts to communicate. Time limit for data to
'      be received is determined in Get Serial Data routine.
'      Func returns 1 if all is ok 0 if CANCEL was received.
'=====
Tried = 0      ' Clear # of times connection was tried
WsComm = 0      ' setup to return failure flag unless success

IF ConnectWs(Ws) = 0 THEN GOTO EndWsComm      ' make connection to Ws, if not successful return 0 now!
' if connection fails then abort this sub

DO      ' Loop up to 5 times to establish communications
    Transmit "[START]" + CRLF$      ' attempt to get HP's attention
    ' -- wait for data from work station --
    St = 0      ' clear time for BgDelay
    Dn = 3      ' delay number unique to this sub!!!
    TimeofLoop! = 2000      ' BgDelay 2 seconds
    DO      ' loop till data is xmitted or time is up
        CALL BgDelay(TimeofLoop!, TimeLeft!, St, Dn)' check for timeout
    LOOP UNTIL (St <= 0 OR DataWaiting)      ' 2 second limit
    ' -- if data is returned from work station --
    IF DataWaiting THEN      ' if data is found
        GetSerialData Return$      ' Read xmitted data
        ' Valid responses are "[REQUEST] (and a time) or [CANCEL]

        IF LEFT$(Return$, 10) = "[REQUEST] " THEN
            TimeRequest# = VAL(MID$(Return$, 11, LEN(Return$)))
            IF TimeRequest# < 2.115E+11 OR TimeRequest# > 2.14E+11 THEN
                Transmit "[ERROR] INVALID DATA RECEIVED >->" + Return$ + "<-<" + CRLF$
                GOTO FailedAttempt      ' which inc's # failed attempts
            END IF
            WStat(Ws).QueTime = TimeRequest#
            Transmit WsMessage$ + CRLF$

```



```

WsComm = 1                ' Successful connection
GOTO EndWsComm            ' Disconnect and Exit function

ELSEIF LEFT$(Return$, 8) = "[CANCEL]" THEN
    WStat(Ws).QueTime = 9E+11    ' This is a long time away
    Transmit "[QUIT]" + CRLF$
    GOTO EndWsComm            ' Disconnect and Exit function
END IF                    ' neither REQUEST or CANCEL rec'd
END IF                    ' if no data is found (upomn switch connection)
FailedAttempt: Tried = Tried + 1    ' inc failed count
LOOP WHILE (Tried < 5)    ' try communications again

EndWsComm:                ' label whereby all must pass to leave
Transmit CHR$(10) + "+++" + CHR$(13)    ' Send Disconnect
ClearInputBuffer          ' just clear the buffer???

END FUNCTION

```


Exhibit 4. HOST PC SOFTWARE: MENU.BAS


```

*****
MENU.BAS
Version 1.20 ~ MENU.BAS is a Microsoft(tm) QuickBASIC v4.5 Module that
contains the procedures used for displaying the operator menu screen.

Written by D.C.Mace, Dept 1940, August 21, 1989
AT&T MicroElectronics, Reading, PA
AT&T Microelectronics - Proprietary. Use Pursuant to Company Instructions
*****

```

```

'/=====/'
DEFINT A-Z          ' All Variables as integers
REM $INCLUDE: 'declare.inc'  ' Declare all procedures

```

```

SUB ClrScr (Mode%, FillChar$)
'-----
' This procedure clears the screen in 2 different ways. The parameters
' are as follows:
'
' mode% - A number indicating which way you want the screen cleared.
'          The number must be in the range of 0 through 14.
' fillChar$ - This is a single character string containing the
'             character you want to clear the screen with. Under
'             most circumstances, this will simply be a space.
'-----

```

```

' The Delay local variable is used here for dummy loops that create a
' very brief pauses of execution at points in the routine that need it,
' particularly in the vertical motion. Change this value to suit the
' speed of your machine.
'/-----/'

```

```

DEFINT A-Z
'---- Clear the screen. Method used is based on the passed Mode% parameter
SELECT CASE Mode

```

```

CASE 1 ' Upper-left corner to lower-right
    fill$ = ""
    FOR x% = 1 TO 25
        fill$ = fill$ + STRING$(3, FillChar$)
        LOCATE x%, 1, 0
        PRINT fill$;
        FOR Y% = 1 TO x%
            LOCATE Y%, x% * 3, 0
            PRINT STRING$(3, FillChar$);
        NEXT Y%
    NEXT x%
    FOR Y% = 1 TO 25
        LOCATE Y%, 78, 0
        PRINT STRING$(3, FillChar$);
    NEXT Y%

```

```

CASE 2 ' Random blocks
    RANDOMIZE TIMER
    REDIM screenGrid%(1 TO 5, 1 TO 10)

    '---- Initialize grid tracking array to all false
    FOR row% = 1 TO 5
        FOR Col% = 1 TO 10
            screenGrid%(row%, Col%) = FALSE
        NEXT Col%
    NEXT row%
    FOR x% = 1 TO 50

```



```

'---- Find a block of the screen that hasn't been blanked yet
ValidBlock% = FALSE
DO
    row% = INT(RND(1) * 5) + 1
    Col% = INT(RND(1) * 10) + 1
    IF screenGrid%(row%, Col%) = FALSE THEN
        ValidBlock% = TRUE
        screenGrid%(row%, Col%) = TRUE
    END IF
LOOP UNTIL ValidBlock%

'---- Blank out the block
FOR i% = ((row% * 5 + 1) - 5) TO ((row% * 5 + 1) - 5) + 4
    LOCATE i%, (Col% * 8 + 1) - 8, 0
    PRINT STRING$(8, FillChar$);
NEXT i%
NEXT x%
CASE ELSE
END SELECT
LOCATE 1, 1, 0
'
END SUB

SUB DisplayEntry (entry$, qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, actionCode%)
'-----
' This procedure is used only by the MakeMenu% Function. It is not meant
' for use on its own. The routine displays the passed menu entry on the
' screen, and highlights the character that proceeds the marker
' character.
'-----
' Parameters are as follows:
'
'   entry$ - The actual text entry to display on the screen
'   qfg% - Foreground color for "Quick Access" key character
'   qbg% - Background color for "Quick Access" key character
'   hfg% - Foreground color for entry at highlight bar
'   hbg% - Background color for entry at highlight bar
'   fg% - Foreground color for normal entry
'   bg% - Background color for normal entry
'   marker$ - The character used in menu entry strings that indicates
'              the next character is a "Quick Access" key.
'   actionCode% - Has value of 1 or 2. 1 indicates that the entry
'                  being displayed is a normal, unhighlighted entry,
'                  thus the "Quick Access" character in the entry will
'                  be highlighted. If 2, "Quick Access" key is not
'                  highlighted, since entry is in highlight bar.
'
' Assumes cursor is already at the right spot to display entry on.
' Display each character until the marker char is found. Print highlighted
' "Quick Access" char if ActionCode% is 1, otherwise print normal "Quick
' Access" char. Then print rest of entry and return to MakeMenu%.
'-----/
DEFINT A-Z
FOR x% = 1 TO LEN(entry$)
    IF MID$(entry$, x%, 1) = Marker$ THEN
        x% = x% + 1
        SELECT CASE actionCode%
            CASE 1
                COLOR qfg%, qbg%

```



```

        CASE 2
            COLOR hfg%, hbg%
        CASE ELSE
            END SELECT
        END IF
        PRINT MID$(entry$, x%, 1);
        IF actionCode% = 2 THEN
            COLOR hfg%, hbg%
        ELSE
            COLOR fg%, bg%
        END IF
    NEXT x%
END SUB

```

SUB ErrorTone (Etone%)

```

'-----
' Error Tone procedure: Make a nice but strange! noise to signal an error
' condition.
'-----

```

DEFINT A-Z

SELECT CASE Etone%

'---- General errors

CASE 1

PLAY "T120 MB L16 O1 AD"

'---- Printer errors

CASE 2

PLAY "MB T250 O2 L16 GGG L10 E- P8 L16 FFF L10 D"

'---- Illegal text input

CASE 3

PLAY "L1603EC"

END SELECT

END SUB

FUNCTION MakeMenu% (Choice\$(), NumOfChoices%, Justify\$, LeftColumn%, RightColumn%, row%, Marker\$, fg%, bg%, hfg%, hbg%, qfg%, qbg%, HelpText\$(), CurrentLocation%)

```

'-----
' The MakeMenu FUNCTION displays a menu list on the screen and allows
' the user to move a scrolling selection bar to highlight the entry of
' their choice. Selection is made by hitting the <ENTER> key. Other
' allowable keys include <HOME> or <PG UP> to move to the first menu entry,
' and <END> or <PG DN> to move to the last entry. Scroll bar wraps from top
' to bottom and bottom to top. The function returns, as an Integer
' value, the position of the entry in the list of the user's selection.
' For example, if the user selected the third item in a list of eight,
' the function would return a value of three.
'-----

```

' Parameters for this FUNCTION are:

```

' choice$() - An array of strings that contains the actual menu
'             entries. Example: Choice$(1) = "Menu selection 1".
'             Strings must be 78 characters or less in length.
' numOfChoices% - The number of menu choices available. The same as
'                 the number of elements in Choice$(). Allowable
'                 range is 1 through 25.
' justify$ - This string will contain a single letter, either an L, C,
'            or a R. L means left-justify the menu entries. C means
'            center them with respect to the left and right sides of

```



```

'         the menu (see LeftColumn and RightColumn parameters below)
'         and an R means right-justify the menu entries.
' LeftColumn - A numerical value containing the left-most column on
'               which menu entries will be displayed. Allowable range
'               is 1 through 76.
' RightColumn - A numerical value containing the right-most column on
'               which menu entries will be displayed. Allowable range
'               is 5 through 80.
' Row% - A numeric value containing the first row on which to display
'         menu entries. Allowable range is 1 through 24.
' marker$ - The character used in the menu entry strings that indicates
'           the next character is a "Quick Access" key.
' fg% - The foreground color of normal menu entries. Allowable range
'       is 0 to 15.
' bg% - The background color of normal menu entries. Allowable range
'       is 0 to 7.
' hfg% - The foreground color of the highlighted menu entry. Allowable
'        range is 0 to 15.
' hbg% - The background color of the highlighted menu entry. Allowable
'        range is 0 to 7.
' qfg% - The foreground color of the Quick Access keys. Allowable
'        range is 0 to 15.
' qbg% - The background color of the Quick Access keys. Allowable
'        range is 0 to 7.
'/=====/
DEFINT A-Z

'---- Set local variables - extended scan codes for keypad keys.
up$ = CHR$(0) + CHR$(72)
down$ = CHR$(0) + CHR$(80)
enter$ = CHR$(13)
home$ = CHR$(0) + CHR$(71)
end$ = CHR$(0) + CHR$(79)
pgUp$ = CHR$(0) + CHR$(73)
pgDn$ = CHR$(0) + CHR$(81)

'---- Set type of justification to uppercase
Justify$ = UCASE$(Justify$)
width% = (RightColumn - LeftColumn - 1)

'---- Check for out-of-bounds parameters. If any are out of range,
'       quit the function.
IF NumOfChoices% < 2 OR NumOfChoices% > 25 THEN EXIT FUNCTION
IF Justify$ <> "C" AND Justify$ <> "L" AND Justify$ <> "R" THEN EXIT FUNCTION
IF LeftColumn < 1 OR RightColumn > 80 THEN EXIT FUNCTION
IF row% < 1 OR row% > 24 THEN EXIT FUNCTION

'---- Calculate the array of character identifiers
REDIM charID(NumOfChoices%) AS STRING * 1
FOR x% = 1 TO NumOfChoices%
    FOR Y% = 1 TO LEN(Choice$(x%))
        IF MID$(Choice$(x%), Y%, 1) = Marker$ THEN
            charID(x%) = UCASE$(MID$(Choice$(x%), Y% + 1, 1))
        EXIT FOR
    END IF
NEXT Y%
NEXT x%

'---- Calculate length of longest menu choice and store value in ChoiceLen%
choiceLen% = 0

```



```

FOR x% = 1 TO NumOfChoices%
    IF LEN(Choice$(x%)) > choiceLen% THEN
        choiceLen% = LEN(Choice$(x%))
    END IF
NEXT x%
choiceLen% = choiceLen% - 1

'---- Determine left-most column to display highlight bar on
Col = (((RightColumn - LeftColumn - 1) - choiceLen%) / 2) + LeftColumn

'---- Print menu choices to screen based on the type of Justification
'    selected (Center, Left, Right).
COLOR fg, bg
Colr = bg * 16 + fg
SELECT CASE Justify$
    CASE "C"
        FOR x% = 1 TO NumOfChoices%
            xCol% = ((width% - (LEN(Choice$(x%))) - 1) \ 2 + LeftColumn) + 1
            LOCATE (row% - 1) + x%, LeftColumn - 1, 0
            Qprint0 SPACE$(choiceLen% + 2), Colr
            LOCATE (row% - 1) + x%, xCol%, 0
            DisplayEntry Choice$(x%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 1
        NEXT x%
    CASE "R"
        FOR x% = 1 TO NumOfChoices%
            LOCATE (row% - 1) + x%, LeftColumn - 1, 0
            Qprint0 SPACE$(choiceLen% + 2), Colr
            LOCATE (row% - 1) + x%, (RightColumn - LEN(Choice$(x%)))
            DisplayEntry Choice$(x%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 1
        NEXT x%
    CASE "L"
        FOR x% = 1 TO NumOfChoices%
            LOCATE (row% - 1) + x%, LeftColumn - 1, 0
            Qprint0 SPACE$(choiceLen% + 2), Colr
            LOCATE (row% - 1) + x%, LeftColumn, 0
            DisplayEntry Choice$(x%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 1
        NEXT x%
END SELECT

'---- Highlight the first entry in the list. Must take into account the
'    justification type.
SELECT CASE CurrentLocation%
    ' Determine which menu choice
    ' to highlight first.
    CASE IS <= 0
        CurrentLocation% = 1
    CASE IS > NumOfChoices%
        CurrentLocation% = NumOfChoices%
    CASE ELSE
END SELECT

COLOR hfg, hbg
Colr = hbg * 16 + hfg
LOCATE row% - 1 + CurrentLocation, LeftColumn - 1: Qprint0 SPACE$(choiceLen% + 2), Colr

SELECT CASE Justify$
    CASE "C"
        xCol% = ((width% - (LEN(Choice$(CurrentLocation%))) - 1) \ 2 + LeftColumn) + 1
        LOCATE (row% - 1 + CurrentLocation%), xCol%, 0
        DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 2
    CASE "R"
        LOCATE (row% - 1) + CurrentLocation%, (RightColumn - LEN(Choice$(CurrentLocation%)))

```



```

        DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 2
CASE "L"
    LOCATE (row% - 1) + CurrentLocation%, LeftColumn
    DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 2
END SELECT

'---- Read keystrokes and change the highlighted entry appropriately
'    and Display Instructions on line 25.
OffCenter " To Select Item, use Cursor Keys then Press [Enter] or Press Highlighted Letter.", 24, 1, 80, 15, 0

Help$ = HelpText$(CurrentLocation%)      ' Display help on 25th line
LOCATE 25, 1: Qprint0 STRING$(80, " "), 113 ' Clear 25th line
OffCenter Help$, 25, 1, 80, 1, 7

StartScanning:
    WHILE INKEY$ > ""                    ' Clear keyboard buffer so
    WEND                                ' cursor stops suddenly.
    '---- Read keystrokes
    key$ = ""

    start! = TIMER                      ' Start timer for ScreenBlank
    WaitTillBlank = 180                 ' Wait 180 seconds before blank

WHILE key$ = ""
    key$ = UCASE$(INKEY$)

'++++ Display Time & Date on the menu screen ++++
    hr = VAL(LEFT$(TIME$, 2))

    '---- Time is displayed in 12 hr format with Am/Pm.
    IF hr = 0 THEN                      ' It is between midnight and 1 am
        hr = 12
        AMorPM$ = " am"
    ELSEIF hr > 12 THEN                 ' After 1:00 pm
        hr = hr - 12                   ' Conversion to 12 hr. format.
        AMorPM$ = " pm"
    ELSEIF hr = 12 THEN                 ' Between noon and 1:00 pm
        AMorPM$ = " pm"
    ELSE                                ' Anything else is between 1:00 am
        AMorPM$ = " am"                ' and noon.
    END IF

    '---- Adjust column that time is printed on.
    SELECT CASE hr
        CASE 1 TO 9
            timecol = 62                ' IF Hr is 1 thru 9.
            LOCATE 5, 61                ' Clear leading space when hr
            Qprint0 " ", 68             ' switches from 12 to 1.
        CASE ELSE
            timecol = 61                ' IF Hr is 10,11,12
    END SELECT

    '---- Display Time
    CurTime$ = STR$(hr) + RIGHT$(TIME$, 6) + AMorPM$
    LOCATE 5, timecol
    Qprint0 CurTime$ + " ", 48

    '---- Build & Display date string
    Month$ = MonthName$(VAL(LEFT$(DATE$, 2)))
    CurrentDate$ = Month$ + " " + MID$(DATE$, 4, 2) + ", " + MID$(DATE$, 7, 4)

```



```
LOCATE 5, 7
Qprint0 " " + CurrentDate$ + " ", 48
```

```
'++++ ScreenBlank +++++
```

```
Finish! = TIMER
```

```
IF ABS(Finish! - start!) > WaitTillBlank THEN ' Find elapsed time
BlankKey$ = ScreenBlank$(15) ' Blank screen & move message every 15 seconds
EXIT FUNCTION
END IF
```

```
WEND
```

```
SELECT CASE key$
```

```
'---- Legal Movement
```

```
CASE up$, down$, home$, end$, pgUp$, pgDn$
```

```
'---- Restore old highlighted choice to normal colors
```

```
COLOR fg, bg
```

```
Colr = bg * 16 + fg
```

```
LOCATE (row% - 1) + CurrentLocation%, LeftColumn - 1: Qprint0 SPACE$(choiceLen% + 2), bg * 16 + fg
```

```
SELECT CASE Justify$
```

```
CASE "C"
```

```
xCol% = ((width% - (LEN(Choice$(CurrentLocation%))) - 1) \ 2 + LeftColumn) + 1
```

```
LOCATE (row% - 1 + CurrentLocation%), xCol%, 0
```

```
DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 1
```

```
CASE "R"
```

```
LOCATE (row% - 1) + CurrentLocation%, (RightColumn - LEN(Choice$(CurrentLocation%)))
```

```
DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 1
```

```
CASE "L"
```

```
LOCATE (row% - 1) + CurrentLocation%, LeftColumn, 0
```

```
DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 1
```

```
END SELECT
```

```
CASE CHR$(32) TO CHR$(127)
```

```
' If valid KEY code, then restore old entry
```

```
FOR x% = 1 TO NumOfChoices%
```

```
IF key$ = charID(x%) THEN
```

```
COLOR fg, bg
```

```
Colr = bg * 16 + fg
```

```
LOCATE (row% - 1) + CurrentLocation%, LeftColumn - 1: Qprint0 SPACE$(choiceLen% + 2), bg * 16 + fg
```

```
SELECT CASE Justify$
```

```
CASE "C"
```

```
xCol% = ((width% - (LEN(Choice$(CurrentLocation%))) - 1) \ 2 + LeftColumn) + 1
```

```
LOCATE (row% - 1 + CurrentLocation%), xCol%, 0
```

```
DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 1
```

```
EXIT FOR
```

```
CASE "R"
```

```
LOCATE (row% - 1) + CurrentLocation%, (RightColumn - LEN(Choice$(CurrentLocation%)))
```

```
DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 1
```

```
EXIT FOR
```

```
CASE "L"
```

```
LOCATE (row% - 1) + CurrentLocation%, LeftColumn, 0
```

```
DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 1
```

```
EXIT FOR
```

```
END SELECT
```

```
END IF
```

```
NEXT x%
```

```
CASE ELSE
```

```
' Do nothing
```

```
END SELECT
```

```
'---- Update our highlight bar's location based on which key was hit
```



```

SELECT CASE key$
    CASE up$

'---- Set new currentLocation%
    IF CurrentLocation% = 1 THEN
        CurrentLocation% = NumOfChoices%
    ELSE
        CurrentLocation% = CurrentLocation% - 1
    END IF
CASE down$

'---- Set New currentLocation%
    IF CurrentLocation% = NumOfChoices% THEN
        CurrentLocation% = 1
    ELSE
        CurrentLocation% = CurrentLocation% + 1
    END IF
CASE enter$

'---- Set MakeMenu to highlighted selection and exit
    MakeMenu% = CurrentLocation%

'---- Instead of using exitCode to beak out of this, we have to
'    use EXIT FUNCTION, or it never quits.
    EXIT FUNCTION
CASE home$, pgUp$

'---- Set New currentLocation%
    CurrentLocation% = 1
CASE end$, pgDn$

'---- Set New currentLocation%
    CurrentLocation% = NumOfChoices%
CASE CHR$(32) TO CHR$(127)

'---- Check for "Quick Access" codes
    validEntry% = FALSE
    FOR x% = 1 TO NumOfChoices%
        IF key$ = charID(x%) THEN
            MakeMenu% = x%
            CurrentLocation% = x%
            validEntry% = TRUE
        END IF
    NEXT x%
    IF validEntry% = FALSE THEN
        Oops
    END IF
CASE ELSE
    ' Do nothing
END SELECT

Help$ = HelpText$(CurrentLocation%)
' Display help on 25th line for
' each menu choice.

LOCATE 25, 1: Qprint0 STRING$(80, " "), 113
' Clear 25th line
OffCenter Help$, 25, 1, 80, 1, 7

'---- Highlight the entry indicated by CurrentLocation%
SELECT CASE key$
    CASE up$, down$, home$, end$, pgUp$, pgDn$

'---- Highlight new choice

```



```

COLOR hfg, hbg
Colr = hbg * 16 + hfg
LOCATE (row% - 1) + CurrentLocation%, LeftColumn - 1: Qprint0 SPACE$(choiceLen% + 2), hbg * 16 + hfg
SELECT CASE Justify$
    CASE "C"
        xCol% = ((width% - (LEN(Choice$(CurrentLocation%))) - 1) \ 2 + LeftColumn) + 1
        LOCATE (row% - 1 + CurrentLocation%), xCol%, 0
        DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 2
    CASE "R"
        LOCATE (row% - 1) + CurrentLocation%, (RightColumn - LEN(Choice$(CurrentLocation%)))
        DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 2
    CASE "L"
        LOCATE (row% - 1) + CurrentLocation%, LeftColumn, 0
        DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 2
END SELECT
CASE CHR$(32) TO CHR$(127)
    FOR x% = 1 TO NumOfChoices%
        IF key$ = charID(x%) THEN
            '---- Highlight new choice
            COLOR hfg, hbg
            Colr = hbg * 16 + hfg
            LOCATE (row% - 1) + CurrentLocation%, LeftColumn - 1: Qprint0 SPACE$(choiceLen% + 2), hbg * 16 +
hfg

            SELECT CASE Justify$
                CASE "C"
                    xCol% = ((width% - (LEN(Choice$(CurrentLocation%))) - 1) \ 2 + LeftColumn) + 1
                    LOCATE (row% - 1 + CurrentLocation%), xCol%, 0
                    DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 2
                    EXIT FUNCTION
                CASE "R"
                    LOCATE (row% - 1) + CurrentLocation%, (RightColumn - LEN(Choice$(CurrentLocation%)))
                    DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 2
                    EXIT FUNCTION
                CASE "L"
                    LOCATE (row% - 1) + CurrentLocation%, LeftColumn, 0
                    DisplayEntry Choice$(CurrentLocation%), qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, 2
                    EXIT FUNCTION
            END SELECT
        END IF
    NEXT x%
CASE ELSE
    ' Do nothing
END SELECT

GOTO StartScanning
END FUNCTION

```

```

SUB MakeWindow (TopRow%, LeftCol%, BotRow%, RightCol%, ForeColor%, BackColor%, windowType%, frameType%, shadowColor%, exp
lodeType%, Label$)

```

```

'-----
' This MakeWindow procedure draws windows on the screen. The kinds of
' windows you can make is quite varied. There are 4ss window types, three
' different frame types, windows can have shadows or not, you can "explode"
' them onto the screen, and even place labels on them. The parameters for
' MakeWindow are as follows:

```

```

'
'   topRow% -   This is an integer value containing the top-most row of
'               the window. Allowable range is 1 through 22.
'   leftCol% -  This is an integer value containing the left-most side

```



```

' of the window. Allowable range is 1 to 79.
' botRow% - This is an integer value containing the bottom-most row
' of the window. Allowable range is 2 through 23.
' rightCol% - This is an integer value containing the right-most row
' of the window. Allowable range is 2 through 80.
' foreColor% - This is the foreground color of the window. Allowable
' range is 0 through 15.
' backColor% - This is the background color of the window. Allowable
' range is 0 through 7.
' windowType% - This is an integer value containing the type of window
' desired. Allowable range is 0.
' frameType% - This is an integer value containing the type of frame
' you want your window to have. Allowable range is 0,
' 1, & 3.
' shadowColor% - This is an integer value containing the color of the
' shadow for your window. If you desire no shadow at
' all, use a value of -1. Allowable range is -1 through
' 15.
' explodeType% - This is a integer value that indicates how you want
' your window to be placed on the screen. A value of 0
' display it normally, top to bottom. A value of 1
' means explode it onto the screen using auto mode.
' label$ - This is a string used to label your window. It is placed
' along the top line of your window, framed by brackets.
' A string of zero length ("" ) means don't display any label.
' Allowable string length is equal to (RightCol - LeftCol) - 4
' -----

```

```

DEFINT A-Z

```

```

'----- Setup line$ as a dynamic array that can ReDIMensioned. Line$()
' will contain the actual character strings that make up our window.
REDIM line$(24)

```

```

'----- Initialize local variables
part1 = 0: part2 = 0: numLines = 0

```

```

'----- Check all passed values for validity and set defaults
numLines = 0

```

```

IF TopRow < 1 THEN TopRow = 1: IF TopRow > 22 THEN TopRow = 22
IF BotRow < 2 THEN BotRow = 2: IF BotRow > 25 THEN BotRow = 25
IF RightCol < 2 THEN RightCol = 2: IF RightCol > 80 THEN RightCol = 80
IF LeftCol < 1 THEN LeftCol = 1: IF LeftCol > 79 THEN LeftCol = 79

```

```

IF ForeColor% < 0 OR ForeColor% > 15 THEN ForeColor% = 7
IF BackColor% < 0 OR BackColor% > 7 THEN BackColor% = 0

```

```

IF windowType% < 0 OR windowType% > 3 THEN windowType% = 0
IF frameType% < 0 OR frameType% > 3 THEN frameType% = 0
IF shadowColor% > 15 THEN shadowColor% = 0
IF explodeType% < 0 OR explodeType% > 3 THEN explodeType% = 0

```

```

IF LEN(Label$) > ((RightCol - LeftCol) - 4) THEN Label$ = ""

```

```

'----- Setup graphics characters to use based on FrameType%

```

```

SELECT CASE frameType%
CASE 0 ' All lines SINGLE
    urc$ = CHR$(191): ULC$ = CHR$(218): llc$ = CHR$(192): LRC$ = CHR$(217)
    ver$ = CHR$(179): hor$ = CHR$(196)
    vtl$ = CHR$(195): vtr$ = CHR$(180)
    htt$ = CHR$(194): htb$ = CHR$(193)

```



```
crs$ = CHR$(197): blk$ = CHR$(219)
lbl$ = CHR$(180): lbr$ = CHR$(195)
```

```
CASE 1 ' All lines DOUBLE
```

```
urc$ = CHR$(187): ULC$ = CHR$(201): llc$ = CHR$(200): LRC$ = CHR$(188)
ver$ = CHR$(186): hor$ = CHR$(205)
vtl$ = CHR$(204): vtr$ = CHR$(185)
htt$ = CHR$(203): htb$ = CHR$(202)
crs$ = CHR$(206): blk$ = CHR$(219)
lbl$ = CHR$(181): lbr$ = CHR$(198)
```

```
CASE 3 ' Horizontals DOUBLE / Verticals SINGLE
```

```
urc$ = CHR$(184): ULC$ = CHR$(213): llc$ = CHR$(212): LRC$ = CHR$(190)
ver$ = CHR$(179): hor$ = CHR$(205)
vtl$ = CHR$(198): vtr$ = CHR$(181)
htt$ = CHR$(209): htb$ = CHR$(207)
crs$ = CHR$(216): blk$ = CHR$(219)
lbl$ = CHR$(181): lbr$ = CHR$(198)
```

```
CASE ELSE ' Shouldn't be an "else" !
```

```
END SELECT
```

```
'---- Calculate the number of lines to be printed and redimension Lines$( )
```

```
numLines = (BotRow - TopRow) + 1
```

```
REDIM line$(numLines)
```

```
'---- Determine ExplodeStep% for explode loop based on ExplodeType%
```

```
SELECT CASE explodeType%
```

```
    CASE 0 ' Exploding Windows OFF
        explodeStep% = 0
```

```
    CASE 1 ' Explode automatic - determine explode ratio
```

```
        explodeStep% = INT((RightCol - LeftCol) / (BotRow - TopRow))
```

```
END SELECT
```

```
'---- Construct the window strings based on WindowType%
```

```
SELECT CASE windowType%
```

```
    CASE 0 ' Regular box, no extra lines
```

```
        line$(1) = ULC$ + STRING$(RightCol - LeftCol - 1, hor$) + urc$
```

```
        FOR x% = 2 TO numLines - 1
```

```
            line$(x%) = ver$ + SPACE$(RightCol - LeftCol - 1) + ver$
```

```
        NEXT x%
```

```
        line$(numLines) = llc$ + STRING$(RightCol - LeftCol - 1, hor$) + LRC$
```

```
    CASE ELSE ' Shouldn't be an "else" !
```

```
END SELECT
```

```
'---- Print the Window, Please! Set colors to those passed to MakeWindow
```

```
Colr = BackColor% * 16 + ForeColor%
```

```
'---- Print the window on the screen, using method based on ExplodeType%
```

```
SELECT CASE explodeType%
```

```
    CASE 0 ' No explosion - just a straight print.
```

```
        FOR x% = 1 TO numLines
```

```
            LOCATE (x% + (TopRow - 1)), LeftCol: Qprint0 line$(x%), Colr
```

```
        NEXT x%
```

```
    CASE 1, 2, 3 ' Explode that window!
```

```
        expX1% = INT(((RightCol - LeftCol) / 2) + LeftCol): expX2% = expX1%
```

```
        expY1% = INT(((BotRow - TopRow) / 2) + TopRow): expY2% = expY1%
```



```

        WHILE (expX1% > LeftCol + 1) OR (expY1% > TopRow + 1)
            IF expX1% > LeftCol THEN expX1% = expX1% - explodeStep%
            IF expX2% < RightCol THEN expX2% = expX2% + explodeStep%
            IF expY1% > TopRow THEN expY1% = expY1% - 1
            IF expY2% < BotRow THEN expY2% = expY2% + 1
            IF expX1% < LeftCol THEN expX1% = LeftCol: expX2% = RightCol
            IF expY1% < TopRow THEN expY1% = TopRow: expY2% = BotRow
            LOCATE expY1%, expX1%: Qprint0 ULC$ + STRING$((expX2% - expX1%) - 1, hor$) + urc$, Colr
            FOR x% = expY1% + 1 TO expY2% - 1
                LOCATE x%, expX1%: Qprint0 ver$ + SPACE$((expX2% - expX1%) - 1) + ver$, Colr
            NEXT x%
            LOCATE expY2%, expX1%: Qprint0 llc$ + STRING$((expX2% - expX1%) - 1, hor$) + LRC$, Colr
        WEND

'---- Print a straight window now, after the explosion effect
        FOR x% = 1 TO numLines
            LOCATE (x% + (TopRow - 1)), LeftCol: Qprint0 line$(x%), Colr
        NEXT x%

        CASE ELSE ' Shouldn't be an "else" !
END SELECT

'---- Add a shadow if required
IF shadowColor% >= 0 THEN

'---- Change colors to ShadowColor%
    Colr = shadowColor%

'---- Define the characters to display for the side/bottom shadow
    sideShadow$ = STRING$(2, 219)
    botShadow$ = STRING$((RightCol - LeftCol), 219)

'---- Print the side shadow
    FOR x% = TopRow + 1 TO BotRow + 1
        LOCATE x%, RightCol + 1: Qprint0 sideShadow$, Colr
    NEXT x%

'---- Print the bottom shadow
    LOCATE BotRow + 1, LeftCol + 2: Qprint0 botShadow$, Colr
END IF

'---- Add the Window Label, if possible. Set the colors to those passed
' to MakeWindow routine.
Colr = BackColor% * 16 + ForeColor%

'---- Add label to window if one was specified
IF Label$ <> "" THEN
    Label$ = lbl$ + Label$ + lbr$
    LOCATE TopRow, LeftCol + 1
    Qprint0 Label$, Colr
END IF
END SUB

SUB OffCenter (st$, row%, LeftCol%, RightCol%, fg%, bg%)
'-----
' This procedure will center the text passed to it on the screen between
' two specified columns. Excellent for centering text in a window
' that itself is not centered in the screen. Parameters are:
'
' st$ - the string to center. Maximum length of string is 80

```



```

'      characters.
'      row% - The row on which the string will be centered. Allowable
'              range is 1 through 25.
'      leftCol% - The left-most column to center the text between.
'                  Allowable range is 1 through 79.
'      rightCol% - The right-most column to center the text between.
'                  Allowable range is 2 through 80.
'      fg% - Foreground color;
'      bg% - Background color
' The formula used in the Qprint0 parameter list will support a legal
' foreground flashing color and also prevent an illegal background color
' from causing unintentional flashing.
'-----
'---- Calculate width available for string
width = (RightCol% - LeftCol%)

'---- If ST$ fits in available width, determine X% for Locate. Otherwise,
'      quit the procedure.
IF LEN(st$) > width THEN
    EXIT SUB
ELSE
    x% = INT(((width - (LEN(st$))) / 2) + LeftCol%) + 1
END IF

'---- Print the string
LOCATE row%, x%, 0: Qprint0 st$, (fg% AND 16) * 8 + ((bg% AND 7) * 16) + (fg% AND 15)
END SUB

SUB Oops
'-----
' This procedure produces a pleasant tone instead of the dull BEEP offered on
' the PC.
'-----
    SOUND 1000, 1: SOUND 1500, 1: SOUND 600, 1
'
END SUB

FUNCTION ScreenBlank$ (BDelay%)
'-----
' Blanks the screen to prevent burn-in and displays a message explaining this,
' until a key is hit. Returns a string value for the key that was pressed.
'
' Usage syntax: BlankKey$ = ScreenBlank$(BDelay%)
'
' BDelay% = Seconds of delay for moving message
' BlankKey$ = Variable that holds the returned keypress
'-----
DEFINT A-Z
    RANDOMIZE TIMER
    COLOR 7, 0: CLS
    GOSUB MoveMessage

    start! = TIMER
    DO
        ' Display message until a keypress.
        key$ = INKEY$
        Finish! = TIMER

        '---- Find elapsed time.
        IF ABS(Finish! - start!) > BDelay% THEN
            start! = TIMER

```



```

        GOSUB MoveMessage
    END IF
LOOP UNTIL LEN(key$)

```

```

ScreenBlank$ = key$
EXIT FUNCTION

```

```

MoveMessage:      ' Move the message to a new place on the screen.

```

```

    CLS
    Col% = INT(RND(1) * 33) + 1      ' Origin 38
    row% = INT(RND(1) * 19) + 1      ' Origin 22
    RCol% = Col% + 46
    fg = 7: bg = 1
    MakeWindow row%, Col%, row% + 3, RCol%, fg, bg, 0, 1, 0, 0, ""
    OffCenter "Screen has been Blanked to Prevent Burn-in.", row% + 1, Col%, RCol%, fg%, bg%
    OffCenter "Hit any key to Return to Menu...", row% + 2, Col%, RCol%, fg%, bg%
    RETURN

```

```

END FUNCTION

```

```

SUB ScreenSave (FirstLine%, lastLine%, scrArray%(), Segment%)

```

```

'-----
' This procedure will save all or a portion of the screen display to an
' integer array.

```

```

' Parameters are as follows:

```

```

'
'   firstLine% - The first line of the display where saving should
'               begin. Top line is 1, bottom is 25.
'   lastLine%  - The last line of the display where saving should
'               end, LastLine% being included.
'   scrArray%() - The array in which the display contents will be
'                 stored. Must be integer, and must be dimensioned
'                 to 3999 (or 4000) elements.

```

```

'-----
'---- Determine the starting address in the video memory (start%). Must use
'   160 for the length of a line, since an attribute byte is stored for each
'   character on the screen (80 characters + 80 attributes = 160)
start% = (FirstLine% - 1) * 160

```

```

'---- Calculate the length of the block of addresses we must retrieve and
'   store (length%). 1 is subtracted since the array starts with element 0.
Length% = (((lastLine% - FirstLine%) + 1) * 160) - 1

```

```

'---- Set the default segment to the video memory segment.
DEF SEG = Segment

```

```

'---- Get information (characters and attributes) from video memory.
FOR i% = 0 TO Length%
    scrArray%(start% + i%) = PEEK(start% + i%)
NEXT i%

```

```

'---- Restore default Segment to QuickBASIC's segment.
DEF SEG

```

```

END SUB

```

```

SUB ScrnRestore (FirstLine%, lastLine%, scrArray%(), Segment)

```

```

'-----

```



```

' This procedure will restore all or a portion of the screen display
' from an integer array.
'
' Parameters are as follows:
'
'     firstLine% - The first line of the display where restore should
'                   begin. Top line is 1, bottom is 25.
'     lastLine%  - The last line of the display where restore should
'                   end, LastLine% being included.
'     scrArray%() - The array in which the display contents will be
'                   restored. Must be integer, and must be dimensioned
'                   to 3999 (or 4000) elements.
'-----
'---- Determine the starting address in the video memory (start%). Must use
'     160 for the length of a line, since an attribute byte is stored for each
'     character on the screen (80 characters + 80 attributes = 160)
start% = (FirstLine% - 1) * 160

'---- Calculate the length of the block of addresses we must restore (length%).
'     1 is subtracted since the array starts with element 0.
Length% = (((lastLine% - FirstLine%) + 1) * 160) - 1

'---- Set the default segment to the video memory segment.
DEF SEG = Segment

'---- Restore information (characters and attributes) to video memory.
FOR i% = 0 TO Length%
    POKE start% + i%, scrArray%(start% + i%)
NEXT i%

'---- Restore default segment to QuickBASIC's segment.
DEF SEG

END SUB

FUNCTION SetDate% (StrDate$) STATIC
'-----
' Set Date Procedure: This procedure validates the entering of the correct
' string format for the date, and sets the CMOS RAM chip date for 286/386
' machines. The DOS date is also updated immediately so this this value can
' be used for the date display. SetDate% is returned as a 0 if invalid, or a
' -1 if valid.
' Passed parameter StrDate$: A string argument in the format mm-dd-yyyy, valid
' range of mm = 1 to 12; dd = 1 to 31; yyyy = 1980 to 1999
'-----
DEFINT A-Z
IF StrDate$ = "" THEN EXIT FUNCTION

'---- Look for the required hyphens in the string argument.
Hyphen1 = INSTR(StrDate$, "-")
Hyphen2 = INSTR(Hyphen1 + 1, StrDate$, "-")

'---- If both hyphens are there, begin analyzing the string.
IF Hyphen1 = 3 AND Hyphen2 = 6 THEN

    '---- Find the three date elements, and validate the string format.
    Month = VAL(LEFT$(StrDate$, 2))
    IF Month < 1 OR Month > 12 THEN Month = FALSE

    Day = VAL(MID$(StrDate$, 4, 2))

```



```

    IF Day < 1 OR Day > 31 THEN Day = FALSE

    Year = VAL(MID$(StrDate$, 7, 4)) - 1900
    IF Year < 80 OR Year > 99 THEN Year = FALSE

ELSE
    SetDate = FALSE
    ErrorTone 1
    EXIT FUNCTION

END IF

'---- If date string validated, write to CMOS and update DOS.
IF Month <> FALSE AND Day <> FALSE AND Year <> FALSE THEN
    SetDate = TRUE

    '---- Write Month.
    OUT &H70, 8
    OUT &H71, VAL("&H" + MID$(STR$(Month), 2)) ' Convert to HEX for CMOS

    '---- Write Day.
    OUT &H70, 7
    OUT &H71, VAL("&H" + MID$(STR$(Day), 2)) ' Convert to HEX for CMOS

    '---- Write Year.
    OUT &H70, 9
    OUT &H71, VAL("&H" + MID$(STR$(Year), 2)) ' Convert to HEX for CMOS

    '---- Update DOS date.
    DATE$ = StrDate$

ELSE
    SetDate = FALSE
    ErrorTone 1

END IF

```

,
 END FUNCTION

```

FUNCTION SetTime% (StrTime$) STATIC
'-----
' Set Time Procedure: This procedure validates the entering of the correct
' string format for the time, and sets the CMOS RAM chip time for 286/386
' machines. The DOS time is also updated immediately so this this value can
' be used for the time display. SetTime% is returned as a 0 if invalid, or a
' -1 if valid.
' Passed parameter StrTime$: A string argument in the format hh-mm-ss, valid
' range of hh = 0 to 23 hrs; mm = 0 to 59 mins; ss = 0 to 59 secs.
'-----
DEFINT A-Z
IF StrTime$ = "" THEN EXIT FUNCTION
TempHrs = TRUE: TempMins = TRUE: TempSecs = TRUE

'---- Look for the required colons in the string argument.
Colon1 = INSTR(StrTime$, ":")
Colon2 = INSTR(Colon1 + 1, StrTime$, ":")

'---- If both colons are there, begin analyzing the time string
IF Colon1 = 3 AND Colon2 = 6 THEN

    '---- Find the three time elements, and validate the string format.
    Hours = VAL(LEFT$(StrTime$, 2))
    IF Hours < 0 OR Hours > 23 THEN TempHrs = FALSE

    Minutes = VAL(MID$(StrTime$, 4, 2))

```



```

IF Minutes < 0 AND Minutes > 59 THEN TempMins = FALSE

Seconds = VAL(MID$(StrTime$, 7, 2))
IF Seconds < 0 AND Minutes > 59 THEN TempSecs = FALSE

'---- If time string validated, write to CMOS and update DOS.
IF TempHrs <> FALSE AND TempMins <> FALSE AND TempSecs <> FALSE THEN
    SetTime = TRUE

    '---- Write Hours.
    OUT &H70, 4
    OUT &H71, VAL("&H" + MID$(StrTime$, 1, 2)) ' Convert to HEX for CMOS

    '---- Write Minutes.
    OUT &H70, 2
    OUT &H71, VAL("&H" + MID$(StrTime$, 4, 2)) ' Convert to HEX for CMOS

    '---- Write Seconds.
    OUT &H70, 0
    OUT &H71, VAL("&H" + MID$(StrTime$, 7, 2)) ' Convert to HEX for CMOS

    '---- Update DOS time.
    TIME$ = StrTime$
END IF
ELSE
    SetTime = FALSE
    ErrorTone 1
END IF
'
END FUNCTION

```


Exhibit 5. HOST PC SOFTWARE: MENU.INC

```
'
                                MENU.INC                                *
' Version 1.00 ~ INCLUDE file for initializing the MENU.BAS Module    *
'                                                                    *
'      Written by B.D.Ehrhart, Dept 2010, May 02, 1990                *
'      AT&T Microelectronics, Reading, PA                            *
' AT&T Microelectronics - Proprietary. Use Pursuant to Company Instructions *
*****
```

'---- REDIM Arrays for Main Menu

REDIM Scr1%(1)

NumOfChoices = 6 ' Number of scroll bars

REDIM Choices\$(NumOfChoices%), HelpText\$(NumOfChoices%) ' For Menu text and Help text

'---- Main Menu Screen - Define text for Scroll bars & Help bar on 25th line.

MenuChoice: ' The letter that follows the ^ character

' is the Highlighted letter.

DATA RAMP UP ^LASER POWER, = (Power Up Laser) [ENTER or L] to Select = [ESC] to Abort

DATA ^BEAM Switch, = [ENTER or B] to Select = [ESC] to Abort

DATA ^AUTO Mode, = [ENTER or A] to Select

DATA Set ^Date,Change the System Date = [ENTER or D] to Select

DATA Set ^Time,Change the System Time = [ENTER or T] to Select

DATA ^Quit to DOS,Quit Program & Return to Operating System = [ENTER or Q] to Select

RESTORE MenuChoice

FOR N = 1 TO NumOfChoices

READ Choices\$(N), HelpText\$(N) ' Fill the Arrays

NEXT N

CurrentLocation = 1 ' Menu choice to highlight first

'---- Draw Menu Screen

DrawScreen:

Marker\$ = "" ' Character to mark highlighted letters

Justify\$ = "L" ' Left justify

TopRow = 8 ' Top row here

BotRow = 15 ' Bottom row here

LeftCol = 23 ' Left column here

RightCol = 57 ' Right column here

fg = 15 ' Hi-White FG

bg = 1 ' Blue BG

hfg = 0 ' Black Highlight FG

hbg = 7 ' White Highlight BG

qfg = 12 ' Lt.Red Quick FG

qbg = 1 ' Blue Quick BG

COLOR 7, 0 ' Color background of menu screen

ClrScr 1, " " ' Use 176,177,or 178 for diff. hues

'---- Draw Title Box

MakeWindow 2, 5, 6, 75, 15, 4, 0, 1, -1, 0, ""

OffCenter "BELL LABS SSTC LASER WELDER HOST COMPUTER", 3, 1, 80, 15, 4

OffCenter VersionStr\$, 4, 1, 80, 15, 4 ' Display Version Number and Date

GetChoices:

MakeWindow TopRow, LeftCol, BotRow, RightCol, fg, bg, 0, 3, 0, 0, ""

Choice = MakeMenu(Choices\$(), NumOfChoices, Justify\$, LeftCol + 2, RightCol - 1, TopRow + 1, Marker\$, fg, bg, hfg, hbg, qfg, qbg, HelpText\$(), CurrentLocation)

Exhibit 6. HOST PC SOFTWARE: DECLARE.INC


```

*****
'                               DECLARE.INC                               *
' Version 1.00 ~ INCLUDE file to declare the FUNCTIONS, SUBprograms, and *
' Variable TYPES for the LUMONICS 701 LASER WELDER HOST COMPUTER         *
' Fixture.                                                                *
'                                                                           *
'      Written by B.D.Ehrhart, Dept 2010, October 17,1990                *
'      AT&T Microelectronics, Reading, PA                                *
' AT&T Microelectronics - Proprietary. Use Pursuant to Company Instructions *
*****

```

'---- TYPE Structure Declarations

```

TYPE WorkStationStatus      ' Data type for each workstation type
    QueStat AS STRING * 11   ' Q status
    ILock AS STRING * 11    ' interlock status
    QueTime AS DOUBLE       ' time rec'd from HP
    QuePos AS INTEGER       ' position # in Q
END TYPE

```

```

TYPE LaserStatus           ' Data type for JK-701 status
    Power AS STRING * 13    ' laser power status
    Remote AS STRING * 12   ' laser in remote
    LReady AS STRING * 12   ' laser ready (warmed up)
    PwrLimit AS STRING * 12 ' power limit
    ETrip AS STRING * 13    ' energy trip status
END TYPE

```

```

TYPE InterfaceBoxStatus    ' Data type for Interface Box status
    Remote AS STRING * 13   ' box in remote
    Air AS STRING * 12     ' air pressure
    Shutter AS STRING * 12  ' shutter open / closed
    SingShot AS STRING * 13 ' single shot fired
END TYPE

```

'---- Define the Register Data Type needed for INTERRUPT. Each element of
' type corresponds to a CPU register. Order is extremely important here!,
' and this must be in the main program, since user-defined data types or
' records are illegal in procedures.

```

TYPE RegType
    ax AS INTEGER
    bx AS INTEGER
    cx AS INTEGER
    dx AS INTEGER
    bp AS INTEGER
    si AS INTEGER
    di AS INTEGER
    flags AS INTEGER
END TYPE

```

'-----
' QuickBASIC FUNCTION and SUBprogram Declarations
'-----

```

'---- LWELD FUNCTIONS and SUBprograms
DECLARE FUNCTION DispWarning% (Warning$())
DECLARE FUNCTION InitWelder% ()
DECLARE FUNCTION PWR% (Y%)
DECLARE FUNCTION RampLaser% (Dir$)
DECLARE FUNCTION SetDate% (StrDate$)
DECLARE FUNCTION SetTime% (StrTime$)

```



```

DECLARE FUNCTION SwitchBeam% (DesiredWorkStation%)
DECLARE SUB AutoWeldMode ()
DECLARE SUB CloseScreen ()
DECLARE SUB DispMsg (W%)
DECLARE SUB DispStat ()
DECLARE SUB ErrorTone (Etone%)
DECLARE SUB Oops ()
DECLARE SUB ReQue ()
DECLARE SUB SweepScreen (Scr%(), E%)
DECLARE SUB UpdateTime ()
DECLARE SUB UpdateStatDisp ()
DECLARE SUB Weldit (WorkStation%)

'+++++++ Subs and Func in LWELDTST (RS232 STUFF)
DECLARE FUNCTION PortStatus% ()
DECLARE SUB Filter (InString%)
DECLARE SUB GetSerialData (R%)

'+++++++ From QB SERIAL.DEC ++++++++
DECLARE SUB OpenComm CDECL ALIAS "_open_comm" (BYVAL Port%, BYVAL Wlen%, BYVAL Parity%, BYVAL Baud%, BYVAL HS%)
DECLARE SUB CloseComm CDECL ALIAS "_close_comm" ()
DECLARE FUNCTION WriteChar% CDECL (BYVAL Ascii%)
DECLARE FUNCTION ReadChar% CDECL ()
DECLARE SUB Transmit CDECL ALIAS "_transmit_string" (addr%)
DECLARE FUNCTION DataWaiting% CDECL ALIAS "_data_waiting" ()
DECLARE SUB ClearInputBuffer CDECL ALIAS "_clear_input_buffer" ()
DECLARE SUB CarrierDetect CDECL ALIAS "_carrier_detect_flag" (BYVAL OnOff%)
DECLARE SUB CDtrap CDECL ALIAS "_trap_mode" (BYVAL OnOff%)
DECLARE FUNCTION CarrierLost% CDECL ALIAS "_carrier_state" ()
DECLARE SUB DTRcontrol CDECL ALIAS "_dtr" (BYVAL OnOff%)

'+++++++ Opening Screen & Menu for Lweld.
'---- QBSCR v1.4 ~ FUNCTION and SUBprograms used:
DECLARE FUNCTION MakeMenu% (Choice$(), NumOfChoices%, Justify$, LeftColumn%, RightColumn%, row%, Marker$, fg%, bg%, hfg%,
    hbg%, qfg%, qbg%, HelpText$(), CurrentLocation%)
DECLARE FUNCTION ScreenBlank$ (BDelay%)
DECLARE SUB ClrScr (Mode%, FillChar%)
DECLARE SUB DisplayEntry (entry$, qfg%, qbg%, hfg%, hbg%, fg%, bg%, Marker$, actionCode%)
DECLARE SUB MakeWindow (TopRow%, LeftCol%, BotRow%, RightCol%, ForeColor%, BackColor%, windowType%, frameType%, shadowCol
    or%, explodeType%, Label%)
DECLARE SUB OffCenter (st$, row%, LeftCol%, RightCol%, fg%, bg%)
DECLARE SUB ScrnRestore (FirstLine%, lastLine%, scrArray%(), Segment%)
DECLARE SUB ScreenSave (FirstLine%, lastLine%, scrArray%(), Segment%)

'---- Crescent 'QuickPak' Professional QuickBASIC SUBprograms

'---- Other SUB and FUNCTION declarations

'---- Crescent Software 'QuickPak' Professional Assembler FUNCTIONS and SUBroutines.
DECLARE FUNCTION MonthName$ (Month%)
DECLARE SUB FillScrn (ULRow%, ULCol%, LRRow%, LRCol%, Colr%, Char%, Page%)
DECLARE SUB MPRestore (ULRow%, ULCol%, LRRow%, LRCol%, OrigWidth%, SEG ArrayEl%)
DECLARE SUB ScrnSave (ULRow%, ULCol%, LRRow%, LRCol%, SEG Element%, Page%)
DECLARE SUB Qprint0 (x$, Colr%)

'--- Other Assembler FUNCTIONS by Doug Mace.
DECLARE FUNCTION Bits% (Number%, Msb%, Lsb%)

'-----
' QuickBASIC 4.x Driver for 16BITBD 'C' Language drivers

```


' January 12 1990 - BDE - 2010

'---- 'C' Language SUBroutine ----- From 16BITBD.OBJ ----

```
DECLARE FUNCTION Enter16% CDECL ALIAS "_enter" (BYVAL ADDRESS%, Dat%)
DECLARE FUNCTION Output16% CDECL ALIAS "_output" (BYVAL ADDRESS%, BYVAL Dat%)
DECLARE SUB Readio CDECL (BYVAL ADDRESS%, BYVAL Register%, Dat%)
DECLARE SUB SetTimeout CDECL ALIAS "_set_timeout" (BYVAL Time%)
DECLARE SUB Writeio CDECL (BYVAL ADDRESS%, BYVAL Register%, BYVAL Dat%)
```

' TIMETEST.BAS

' by Doug Mace - BgDelay modified by Bryan Ehrhart 9-10-90

```
DECLARE FUNCTION Ticks! (HundSecs!)
DECLARE SUB BgDelay (MilliSecs!, MilliSecsToGo!, Status%, DelayNum%)
DECLARE SUB INTERRUPT (Intrum AS INTEGER, Inreg AS RegType, Outreg AS RegType)
```

' Define Variables SHARED with all Procedures and Modules

```
COMMON SHARED ULR%, ULC%, LRR%, LRC%, ExitCode%
DIM SHARED CRLF$ ' Carriage Return Line Feed
    CRLF$ = CHR$(13) + CHR$(10)
DIM SHARED BeamPos AS INTEGER ' beam position
    BeamPos = 4 ' default conditon for Beam Switch
DIM SHARED IboxControlWord AS INTEGER ' data to control Interface box
    IboxControlWord = 0
DIM SHARED IboxStatusWord AS INTEGER ' data read from Interface box
    IboxStatusWord = 0
```

'---- DIMension ARRAYS as DYNAMIC and SHARED with SUBprograms and MODULES

CONST DummyTime = 9E+11 ' This is a long time away

REDIM SHARED WStat(1 TO 4) AS WorkStationStatus 'static

WStat(1).QueTime = DummyTime ' This is a long time away

WStat(2).QueTime = DummyTime ' This is a long time away

WStat(3).QueTime = DummyTime ' This is a long time away

WStat(4).QueTime = DummyTime ' This is a long time away

REDIM SHARED LStat(1) AS LaserStatus ' static

REDIM SHARED IBoxStat(1) AS InterfaceBoxStatus ' static

DIM SHARED Warning\$(5)

DIM SHARED BeamADD(4) AS INTEGER ' The following are for beam swithching
' to a workstation

BeamADD(1) = 128

BeamADD(2) = 256

BeamADD(3) = 384

BeamADD(4) = 0

DIM SHARED Interlock(4) AS INTEGER ' The following values are for WorkStation
' Interlocks when interlock is closed

Interlock(1) = 32

Interlock(2) = 64

Interlock(3) = 128

Interlock(4) = 256

' Define GLOBAL CONSTANTS

CONST ShortDelay! = 150 ' 150 mS delay for serial stuff

CONST TicksPerSec! = 1193180 / 65536 ' 18.2065 clock ticks per second

CONST SecsPerHr! = 60 * 60 ' 3600 secs per hr


```
CONST TicksPerDay! = TicksPerSec! * SecsPerHr! * 24
```

```
CONST FALSE% = 0, TRUE% = NOT FALSE%
```

```
CONST BOARDADD% = &HB000          ' INTERFACE BOARD ADDRESS
```

```
' Status bit values for laser and interface box
```

```
CONST Sync = 1                    ' LASER SYNC PULSE
```

```
CONST LaserReady = 2              ' LASER POWERed UP AND READY
```

```
CONST LaserRemote = 4            ' LASER IN REMOTE (CONTROLABLE CONDITION)
```

```
CONST PowerLimit = 8             ' LASER ENERGY WITHIN SPECIFIED LIMITS
```

```
CONST EnergyTrip = 16            ' ENERGY TOO HIGH CAN'T WELD
```

```
'see Interlock() for 32-256
```

```
CONST Remote = 512               ' REMOTE /LOCAL SWITCH 512 REMOTE 0=LOCAL
```

```
CONST AirPress = 1024           ' AIR PRESSURE SWITCH 1024=OK 0=LOW AIR
```

```
CONST LaserPower = 2048         ' AC POWER TO LASER 1024=ON 0=OFF
```

```
' end of Status bits
```

```
' Control bit values for laser and interface box
```

```
' bits 0-2 not used
```

```
CONST RampUp = 8                ' RAMP UP LASER POWER COMMAND
```

```
CONST ShutterOpen = 16          ' OPEN SHUTTER
```

```
CONST SingleShot = 32           ' SINGLE SHOT COMMAND
```

```
CONST RangeSelect = 64          ' NOT USED
```

```
' see BeamADD for value of bits 128-256 (Beam Select bits)
```

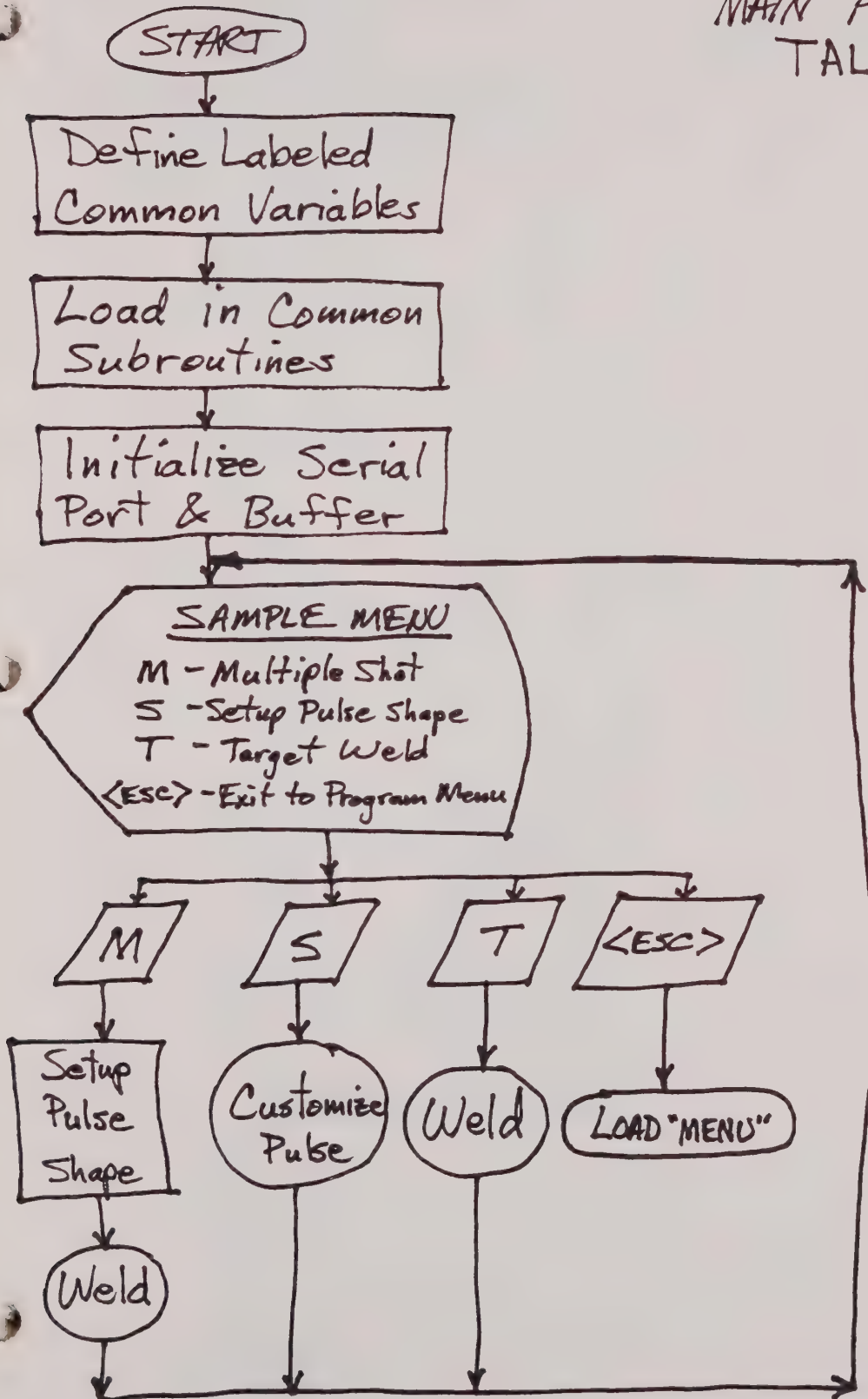
```
' end of Control bits
```


Exhibit 7. PROCESS DIAGRAMS

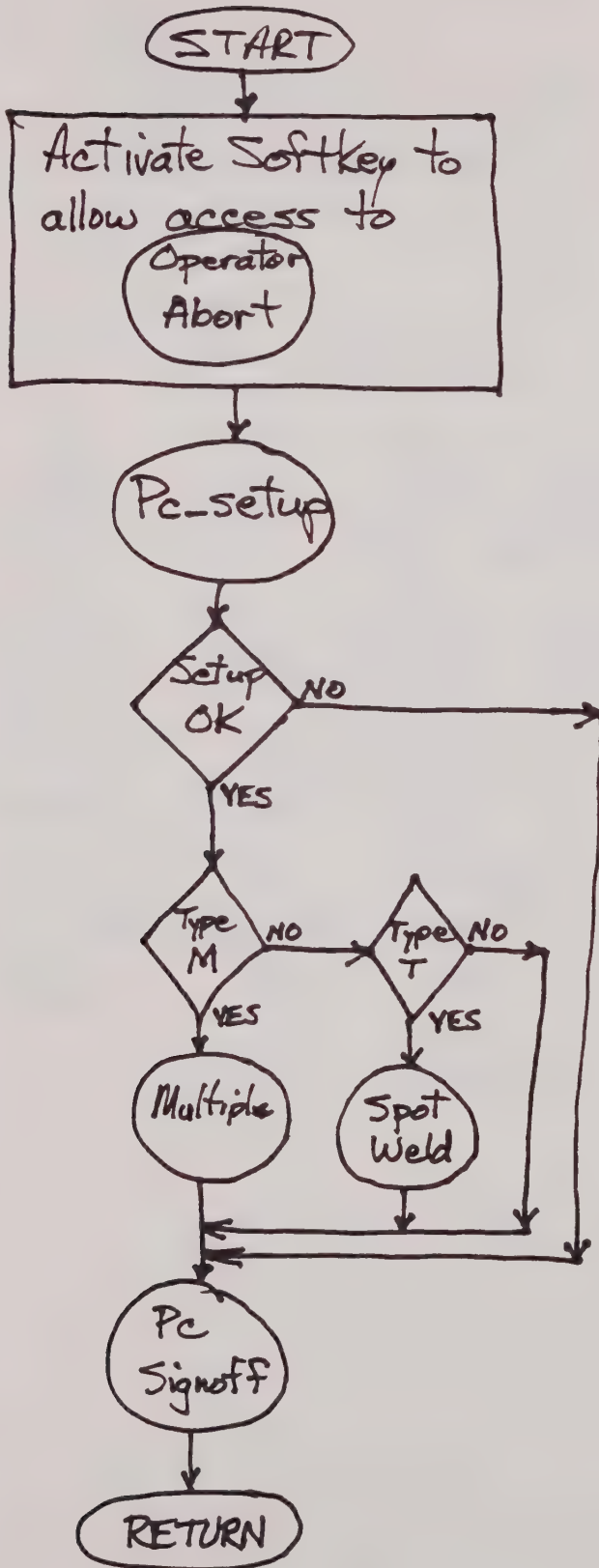
SSTC LUMONICS JK-701 LASER WELDER
SHARED BEAM SYSTEM PROCESS DIAGRAM LISTING

TYPE	ID	NOTES
MAIN PROGRAM	TALKTOPC7	Sample
SUBROUTINE	Weld	Sample
SUBROUTINE	Customize_pulse	
SUBROUTINE	Enter_serial	
SUBROUTINE	Operator_abort	
SUBROUTINE	Pc_release	
SUBROUTINE	Pc_setup	
SUBROUTINE	Pc_signoff	
SUBROUTINE	Setup_serial	
SUBROUTINE	Setup_welder	
SUBROUTINE	Spot_weld	
SUBROUTINE	Wait_to_weld	
DIAGRAM	Communications	Normal Successful

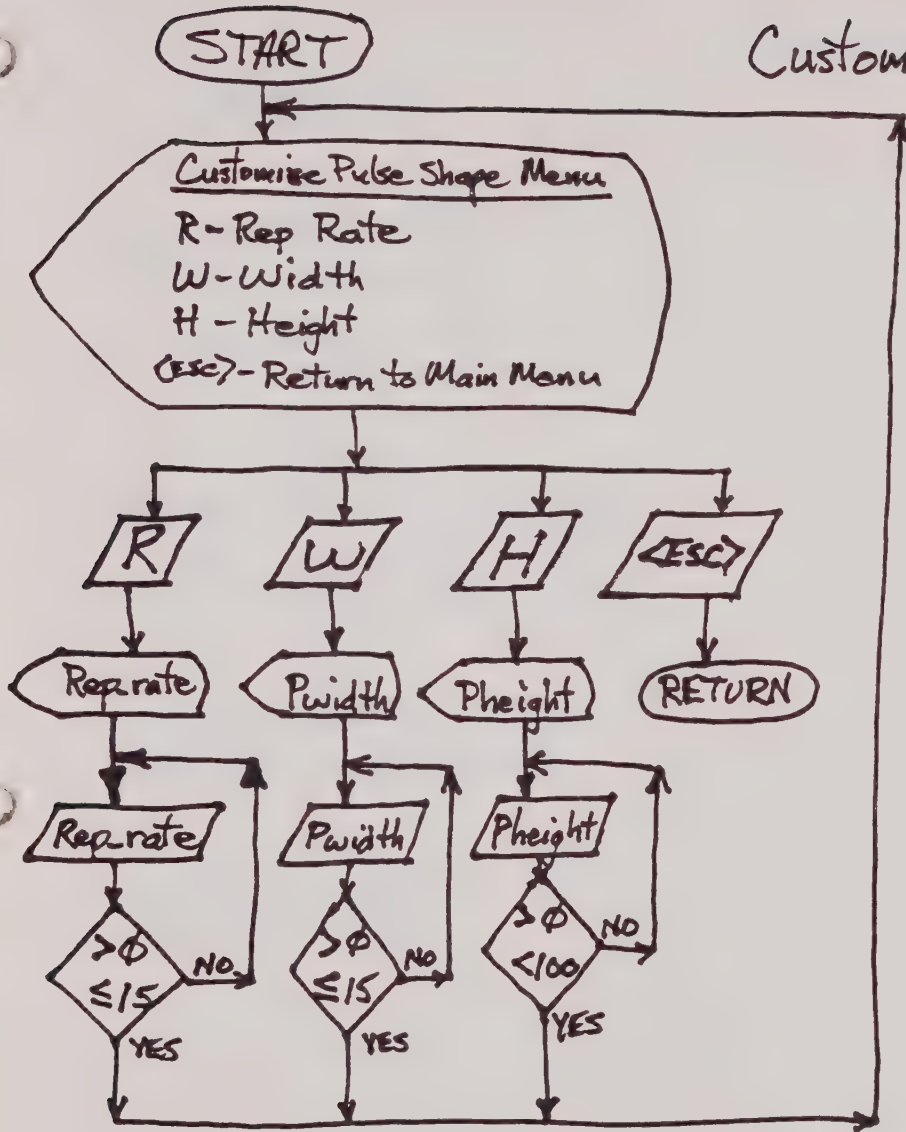
MAIN PROGRAM (SAMPLE TALKTOPC7



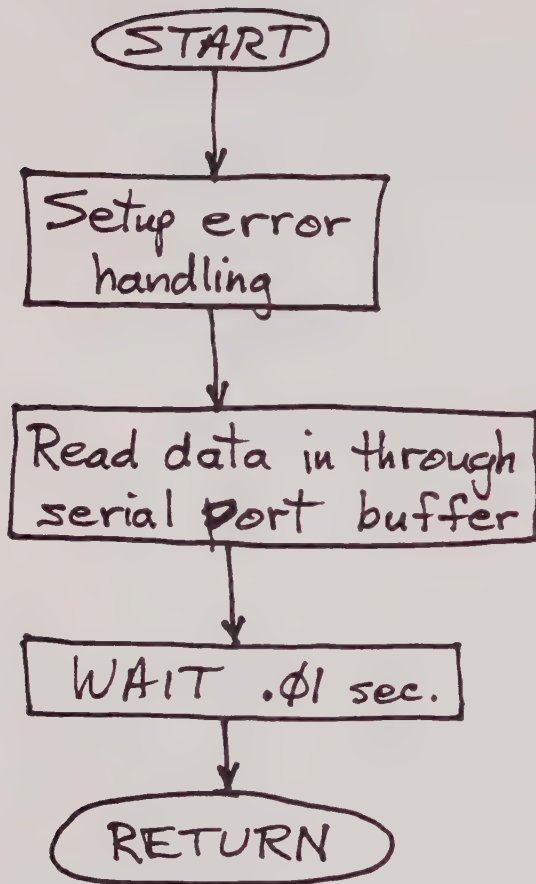
Weld Subroutine (Sample)



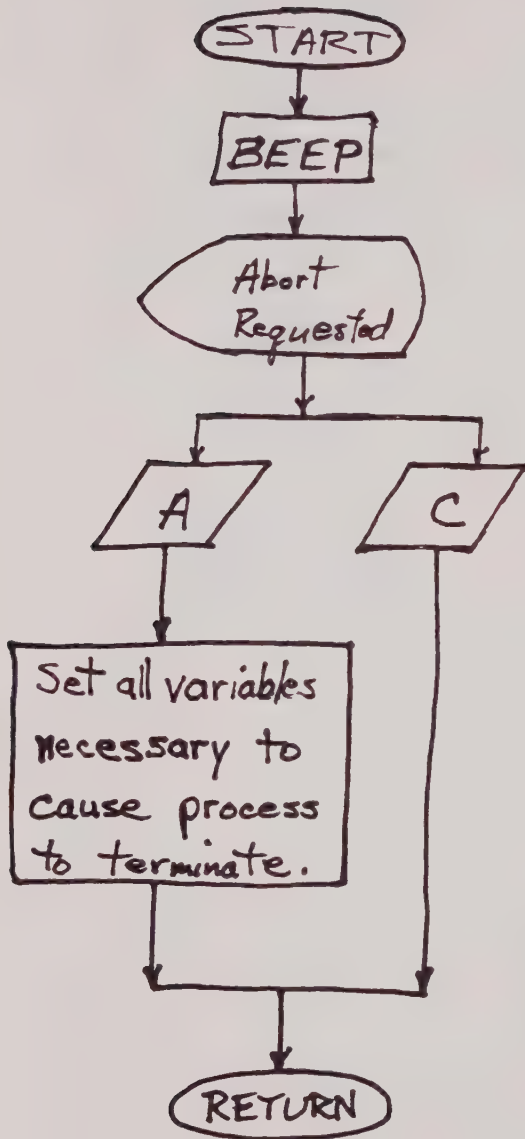
Customize-pulse Subroutine



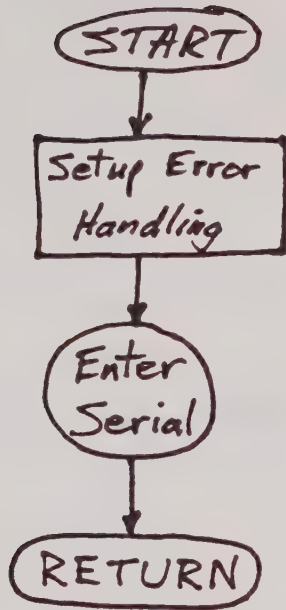
Enter serial Subroutine



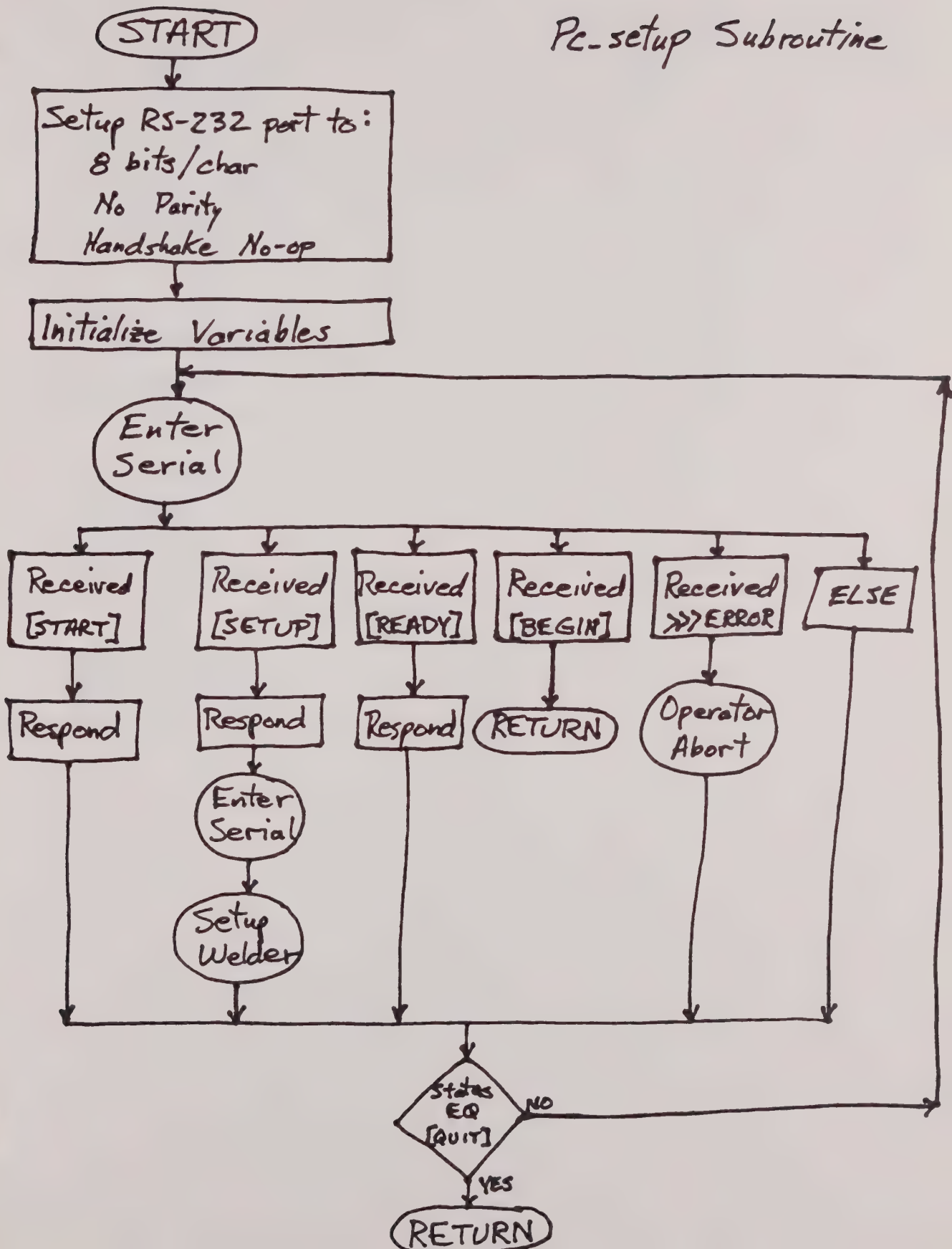
Operator_abort Subroutine



Pc-release Subroutine



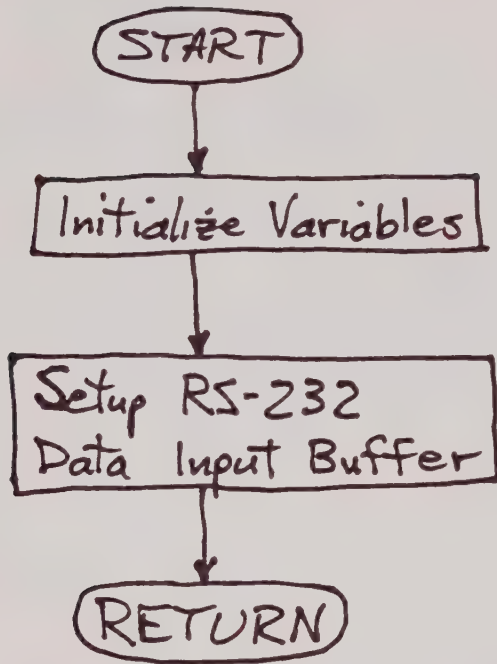
Pc-setup Subroutine



Pc-signoff Subroutine



Setup_serial Subroutine



START

Setup Error Handling

Request Connection thru
Smart Switch to Port 7 (*Luminaries*
Welder)

Setup RS-232 port to:
Xon/Xoff
Even Parity
7 Bits

Enquire Warning Status (EW)

Setup Pulse Shape #1
(Rep-rate, Pulse Height + width)

Enquire Pulse Shape #1 (EX1)

Wait until Laser Power
is stable by Enquiring
Laser Output (EE1)

A

Setup_welder Subroutine

A

ABS
Jocdes-
Target-power
 $\in$
Target-window

Adjust
Pulse
Height

Set Active Pulse
Numbe to be 1 (SO1)

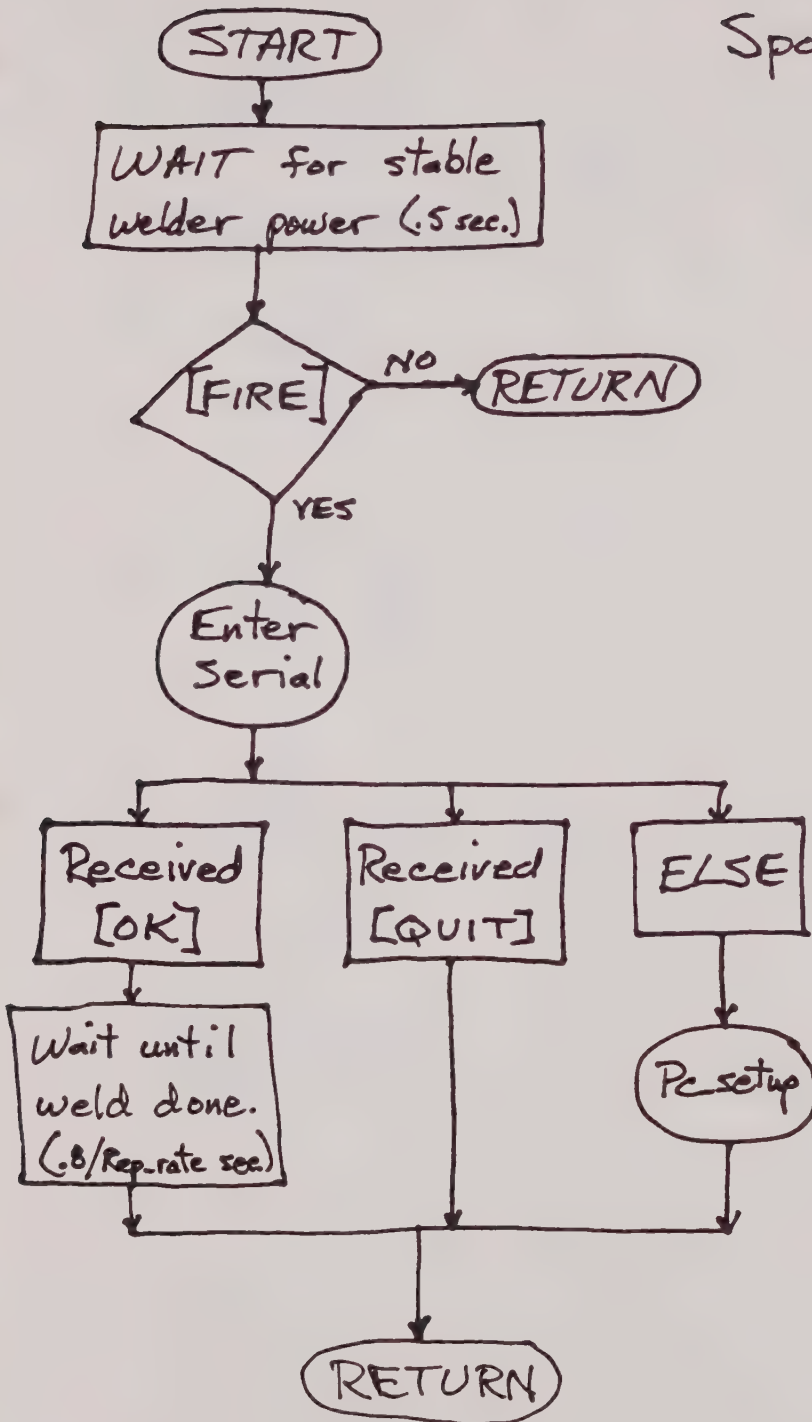
Enquire Active
Pulse Number (EO)

Reset RS-232
Port

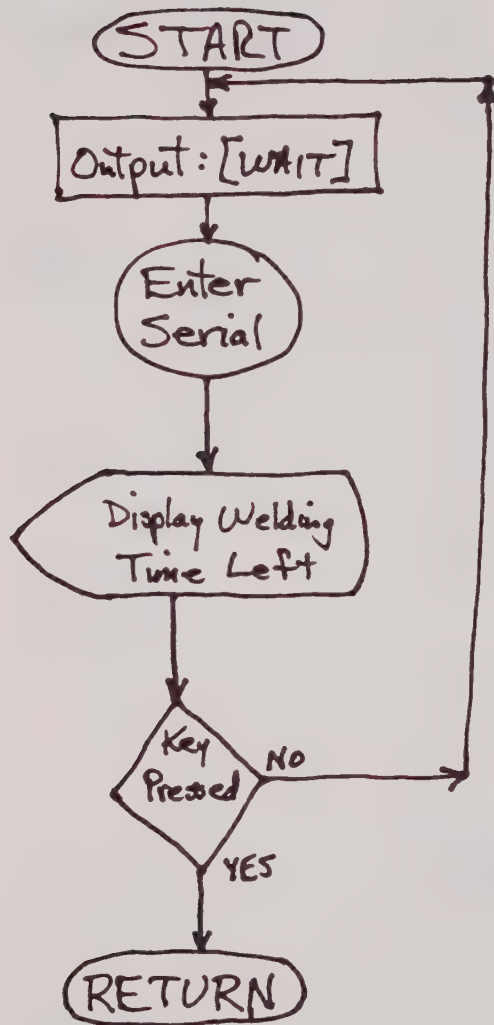
Disconnect from
Smart Switch

RETURN

Spot-weld Subroutine



Wait_to_weld Subroutine



COMMUNICATION DIAGRAM

Normal Successful Transmission

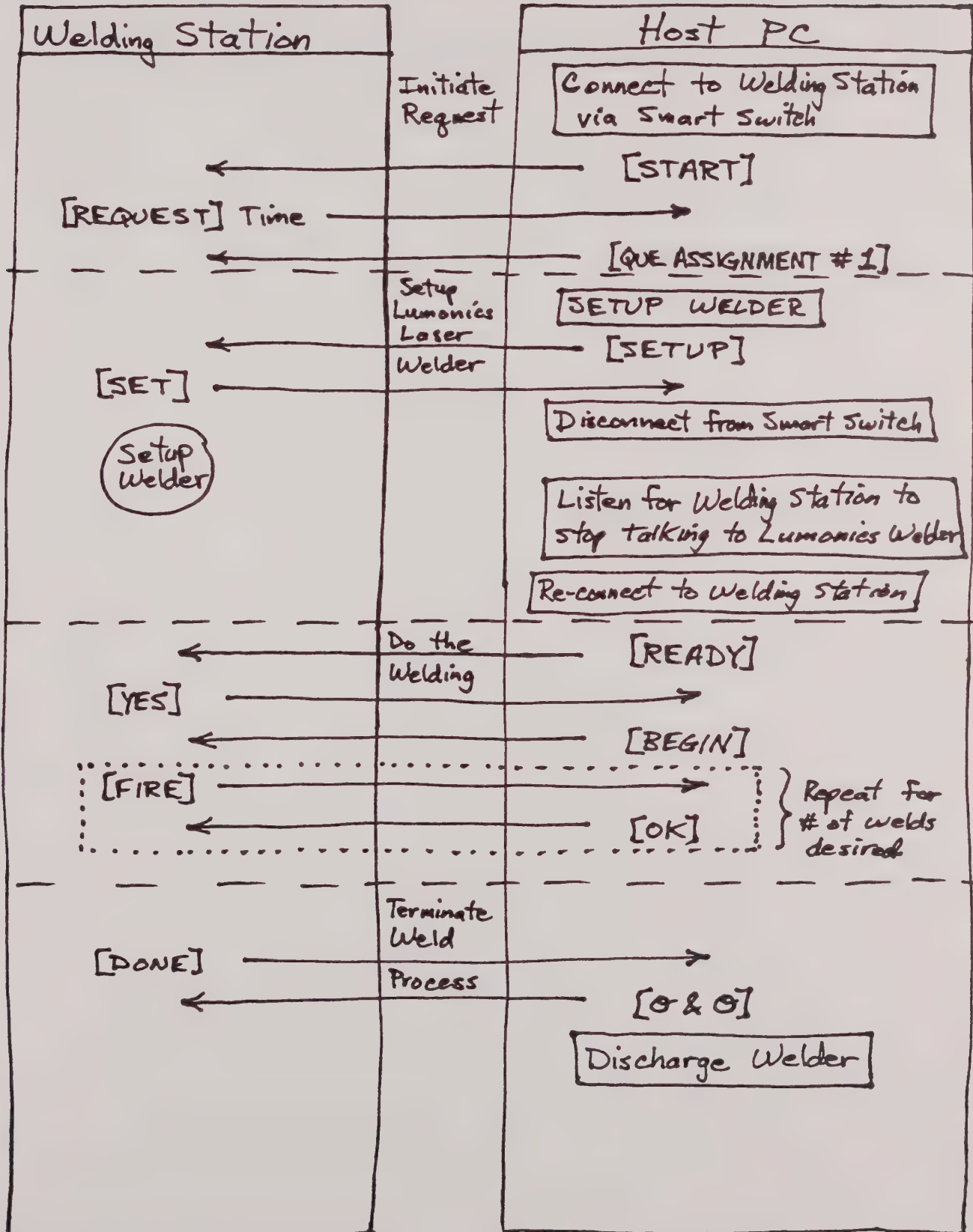


Exhibit 8. SAMPLE COMMUNICATION PRINTOUTS


```

Started Communication w/PC @ 14:11:34
From PC: <<< +++
Bad Message - Wait for another!
From PC: <<< [START] >>>
Welding to PC: <<< [REQUEST] 2.11523148695E+11 >>>
Got Back:[WAIT FOR QUE ASSIGNMENT]
From PC: <<< >>>
From PC: <<< >>>
From PC: <<< +++
Bad Message - Wait for another!
From PC: <<< >>>
From PC: <<< [SETUP] >>>
Setup welder: <<< [SET] >>>
PC disconnected - Ready to setup Welder! ()
About to wake-up Smart Switch!?
Requesting Connection to Port 7
( 0 ) 66 ( 1 ) 32 ( 2 ) 0 ( 3 ) 9600 ( 4 ) 90 ( 5 ) 0 ( 6 ) 120
( 7 ) 4 ( 8 ) 0 ( 9 ) 1 ( 10 ) 110 ( 11 ) 48 ( 12 ) 0 ( 13 )
9600 ( 14 ) 90
Setting:SP1,2.0Hz,1.0mS,90.0% Length: 21
Welder returned:PX1,2.0Hz,1.0mS,90.0%
Welder returned:PX1,2.0Hz,1.0mS,90.0%
Laser Power: .01 -999 1
Laser Power: .01 .01 2
Select Active Pulse!
Setup Welder Status: [OK]
Overrun:Enter_serial:ERROR 167 IN 840 I/O interface status error
From PC: <<< [AB] >>>
Bad Message - Wait for another!
From PC: <<< >>>
From PC: <<< >>>
From PC: <<< [READY] >>>
Ready to weld: <<< [YES] >>>
Welder:[BEGIN] 0 (Welding time left: 4.91 minutes)
Asking for spot weld. (Welding time left: 4.86 minutes)
Spot Weld # 1 Status:[OK] --> Shot Fired!
Signoff Sending: [DONE]
Signoff got from PC:[O&O] --> Communications Terminating!
Pc_release:
Minutes: .216166687012 Seconds: 12.9700012207

```


Started Communication w/PC @ 09:45:20

From PC: <<< >>>

From PC: <<< [START] >>>

Sending to PC: <<< [REQUEST] 2.11524687921E+11 >>>

Got Back:[WAIT FOR QUE ASSIGNMENT]

From PC: <<< >>>

From PC: <<< >>>

>>> PC: <<< +++

Bad Message - Wait for another!

From PC: <<< >>>

From PC: <<< [SETUP] >>>

Setup welder: <<< [SET] >>>

PC disconnected - Ready to setup Welder! ()

About to wake-up Smart Switch!?

Requesting Connection to Port 7

(0) 66	(1) 32	(2) 0	(3) 9600	(4) 90	(5) 0	(6) 248
(7) 4	(8) 0	(9) 1	(10) 103	(11) 48	(12) 0	(13)
9600	(14) 90					

Setting:SP1,2.0Hz,1.0mS,90.0% Length: 21

Welder returned:PX1,2.0Hz,1.0mS,90.0%

Laser Power: 0 -999 1

Laser Power: 4.47 0 2

Laser Power: 6.71 4.47 3

Laser Power: 7.46 6.71 4

Laser Power: 7.61 7.46 5

1 Re-adjusting Power from 7.61 joules to 9.5 (+/- 1)

Pulse Height was 90 % changing to 92 %

Setting:SP1,2.0Hz,1.0mS,92.0% Length: 21

Welder returned:PX1,2.0Hz,1.0mS,92.0%

Laser Power: 7.64 -999 1

Laser Power: 7.66 7.64 2

2 Re-adjusting Power from 7.66 joules to 9.5 (+/- 1)

Pulse Height was 92 % changing to 94 %

Setting:SP1,2.0Hz,1.0mS,94.0% Length: 21

Welder returned:PX1,2.0Hz,1.0mS,94.0%

Welder returned:PX1,2.0Hz,1.0mS,94.0%

Laser Power: 8.22 -999 1

Laser Power: 8.29 8.22 2

3 Re-adjusting Power from 8.29 joules to 9.5 (+/- 1)

Pulse Height was 94 % changing to 96 %

Setting:SP1,2.0Hz,1.0mS,96.0% Length: 21

Welder returned:PX1,2.0Hz,1.0mS,96.0%

Laser Power: 8.44 -999 1

Laser Power: 8.48 8.44 2

4 Re-adjusting Power from 8.48 joules to 9.5 (+/- 1)

Pulse Height was 96 % changing to 98 %

Setting:SP1,2.0Hz,1.0mS,98.0% Length: 21

Welder returned:PX1,2.0Hz,1.0mS,98.0%

Laser Power: 8.59 -999 1

Laser Power: 8.62 8.59 2

Select Active Pulse!

Setup Welder Status: [OK]

Overrun:Enter_serial:ERROR 167 IN 860 I/O interface status error

From PC: <<< [AB] >>> (Welding time left: 1.71 minutes)

Bad Message - Wait for another!

From PC: <<< >>> (Welding time left: 1.71 minutes)

From PC: <<< >>> (Welding time left: 1.64 minutes)

From PC: <<< [READY] >>> (Welding time left: 1.63 minutes)

Ready to weld: <<< [YES] >>>

From PC: <<< [BEGIN] >>> (Welding time left: 1.63 minutes)

Asking for spot weld. (Welding time left: 1.61 minutes)

Spot Weld # 1 Status:[OK] --> Shot Fired!

Signoff Sending: [DONE]

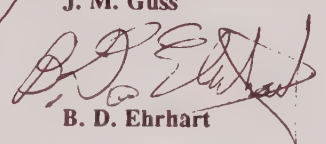
Signoff got from PC:[O&O] --> Communications Terminating!

Pc_release:

Minutes: .521333312988

Seconds: 31.2799987793

STC-52326-JMG/BDE-jmg


J. M. Guss

B. D. Ehrhart

gb /l lweldest.glb lwel11.bas

compile (if prog. name changed, must dg. compile.bat first.)

MC-4/MD-4

STEPPING MOTOR CONTROLLER DRIVER

OPERATOR AND TECHNICAL MANUAL

KLINGER SCIENTIFIC CORPORATION

999 Stewart Avenue ■ Garden City, NY 11530 ■ Tel. No. 516-745-6800

TABLE OF CONTENTS

INTRODUCTION	v
CHAPTER 1: MC4/MD4 System Hookup	1-1
1.1 MD4 to Stage(s) Cable Connection	
1.2 MC4 to MD4 Cable Connection	
1.3 Power Input	
1.4 MC4 Motor Type Programming	
1.5 Motor Type Programming Theory of Operation	
CHAPTER 2: Axis Configuration Switch Settings	2-1
CHAPTER 3: Local Front Panel Operation	3-1
3.1 Front Panel Positioning	
3.2 Manual Low Speed and High Speed Programming	
3.2.1 Setting Low Speed Rate	
3.2.2 Setting High Speed Rate	
3.3 Origin Search	
3.4 Internal Program Selection and Execution	
3.5 Resetting Front Panel Displays	
3.6 Emergency Stop	
3.7 Remote/Local Keyswitch	
CHAPTER 4: IEEE-488 Interface	4-1
4.1 IEEE-488 Connector Pin Definitions	
4.2 MC4 and the IEEE-488	
4.2.1 IEEE-488 Application Program	
4.2.2 Configuring MC4 for IEEE-488	
4.2.3 Testing the IEEE-488 Interface	
4.3 Remote/Local Keyswitch	

SWITCH



CHAPTER 5: RS-232-C Interface	5-1
5.1 RS-232-C Connector for the MC4	
5.2 MC4 and the RS-232-C	
5.2.1 Hardware/Software Handshake	
5.2.2 RS-232-C Application Program	
5.2.3 Configuring MC4 for RS-232-C	
5.2.4 Setting up for RS-232-C	
5.2.5 Testing the RS-232-C Interface	
CHAPTER 6: Automated Control/Command Set Usage	6-1
6.1 Summary of MC4 Commands	
6.2 Theory and Description of Commands	
6.2.1 Principles of Command Execution	
6.2.2 Simultaneous and Interdependent Mode Commands	
6.3 Command Programming Modes	
6.3.1 Real-Time Programming Mode	
6.3.2 Program Entry and Edit Mode	
6.3.3 Program Storage Organization	
6.3.4 Program Execution Mode	
6.3.5 Practice Command Programming Modes	
6.4 Status Report/Request	
6.4.1 Status Format Command	
6.4.2 MC4 Status Response Format	
6.4.3 Status Report Practice	
6.4.4 Error Checking and Reporting	
6.5 Service Request Generation	
6.5.1 Service Request Enable Command	
6.5.2 MC4 Byte Format	
CHAPTER 7: Detailed Description of MC4 Commands	7-1
7.5.1 Summary of MC4 Commands	
7.5.2 Alphabetical Listing and Description of Commands	
CHAPTER 8: MC4 General Purpose Input/Output Connector	8-1
CHAPTER 9: Front Panel Position Display	9-1

CHAPTER 10: MC4 Theory of Operation 10-1

- 10.1 Frequency
- 10.2 Quantity
- 10.3 Direction
- 10.4 Time

CHAPTER 11: Sample Programs 11-1

- 11.1 IEEE-488 Sample Programs
 - 11.1.1 Real Time Devices
 - 11.1.2 National Instruments
 - 11.1.3 HP-85
- 11.2 RS-232-C Sample Programs
 - 11.2.1 IBM-PC: Hardware Handshake
 - 11.2.2 IBM-PC: Software Handshake

CHAPTER 12: Ill-Condition Status Messages: Problems and Solutions 12-1

- 12.1 Operational Problems
- 12.2 IEEE-488 and RS-232-C Communication Problems
- 12.3 Other Problems and Possible Causes
 - 12.3.1 IEEE-488
 - 12.3.2 RS-232-C
 - 12.3.3 Operational

INTRODUCTION

Klinger Scientific's Series MC4 Programmable Stepping Motor Controller provides up to four axes of intelligent stepping motor pulse generation. When connected to a stepping motor driver and a host computer, it forms a compact, high performance unit capable of both basic independent movements and very elaborate repetitive cycles, similar to a robotic system. Different controls, located on the front panel, give the MC4 total autonomy.

Model MC4 is a 4-axis intelligent stepping motor pulse generator. It does not contain any internal power amplifiers to drive stepping motors. It is designed to be used with our MD4 Stepping Motor Power Driver.

The MC4 Programmable Stepping Motor Controller features the following capabilities:

- IEEE-488 and RS-232-C compatible
- flexible command set for automated operation
- manual control via front panel switches and remote control unit
- immediate command execution or command storage in a program buffer for later execution
- simultaneous, sequential, and interdependent command execution
- status reporting while idle and while executing commands
- service request generation at the completion of any move and at the occurrence of a travel limit
- error checking and reporting over the communication channel
- indication of position via LED display and reporting of position over the communication channel
- programmable step pulse rates in either pulses/second or time delay between pulses
- maximum pulse rate: 40,000 pulses/second per axis
- relative and absolute positioning in both the positive and negative directions
- programmable time delay between execution of commands

- variable definition of stepping pulse parameters for succinct program coding
- controlled looping for repetitive execution of commands
- setting and clearing of up to 8 output bits for triggering external devices
- 3 external inputs for synchronization of program execution
- easy editing and listing of internally stored programs
- user program retention in battery back up RAM during power-off state
- 8 kilobytes user program buffer
- front panel selection for execution of up to 9 internally stored programs

Model MD4 Stepping Motor Power Driver is a modular unit capable of driving from one to four stepping motors. This can include any combination of variable reluctance and permanent magnet motors.

Variable reluctance motors are driven in full step mode only. Permanent magnet motors are driven in ministep mode only. A ministep is a fraction of a full step. The MD4 will move a permanent magnet stepping motor one-tenth of a full step for every step pulse it receives from the MC4. For a normally specified 200 step/revolution stepping motor, it now becomes a 2000 step/revolution motor.

HOW THIS MANUAL IS ORGANIZED

The chapter structure of this manual is such that it will guide you through the logical sequence of events leading to a quick and successful use of the MC4/MD4 system. The approach is hands-on; as you read each chapter, you should be doing what it asks you to do.

Chapter 1 describes the proper connection configuration for your system. Here you will connect the MC4 to the MD4 with 15 pin cables. You will also connect the MD4 to the motors/positioning stages with 19 pin cables. When you have made all the connections, you will then program the MC4 from the front panel to tell it what type of motor/stage is connected at each of its axes.

Chapter 2 describes the switch settings which tell the MC4 the parameters of your system. Sample parameters are: encoder/no encoder, 200(48)/2000 step encoder, origin switch on the stage itself, top zero pulse available.

Chapter 3 describes the use and functions of the front panel switches. After you have built and configured your system in chapters 1 and 2, you will use the front panel functions to move and position your stages/motors. You will also use the switches and the displays to program low speed and high speed stepping rates.

Chapters 4 and 5 describe the two communication interfaces used to send commands to the MC4. These commands enable us to duplicate all the functions of the front panel and much more. Chapter 4 describes the IEEE-488 interface in general and, more importantly, as it pertains to the MC4. Chapter 5 describes the RS-232-C serial interface in the same manner. In these chapters you will learn the switch settings on the MC4 necessary to enable your host computer to communicate with the MC4. You will then use the sample programs to send commands to the MC4 and observe the results. Once you have established communication and sent a few rudimentary commands, you will move on to chapter 6.

Chapter 6 describes the theory of operation of MC4 commands. You will learn about the format, the restrictions (how and when they can be used), and the techniques (i.e., grouping of commands) to accomplish some specific tasks. As you learn each new command, you will program it on your host computer and, by observing the MC4 response, you will verify its operation as described in this manual.

Chapter 7 describes all MC4 commands in detail. The commands are listed in alphabetical order for easy location. Each command is described by its function, legal parameters, options and examples of use.

Chapter 8 describes the MC4 General Purpose Input/Output connector. The majority of the input pins of this connector duplicate every switch on the MC4 front panel. By grounding an input pin, the same result is obtained as if you pressed a switch on the front panel of the same designation. Three additional input pins serve as event synchronization, (i.e., the MC4 may be programmed to wait or proceed based on the logic level of any one of these inputs). Eight output pins serve as programmable output bits. The MC4 has two commands that enable you to set and clear any of these bits.

Chapter 9 describes the MC4 front panel display. You will see what the display represents and how it is to be interpreted.

Chapter 10 describes the theory of operation of the MC4.

Chapter 11 presents the sample programs for the IEEE-488 interface and the RS-232-C interface.

Chapter 12 describes the error and status messages available from the MC4. It also describes possible problems you may encounter and their probable cause.

CHAPTER 1: MC4/MD4 SYSTEM HOOKUP

This chapter describes the step by step procedure for properly connecting all the components of your system. It is advised that you physically perform each step as you read it. This will insure proper set-up and no harm to the system. An improper connection may damage the internal boards.

The procedure described assumes a four motor/stage system. If your system will consist of fewer than four stages, go no further than necessary.

1.1 MD4 TO STAGE(S) CABLE CONNECTION

The cable used may be 2 meters, 5 meters, or a custom length. It is terminated with a round 19 pin female connector on one side and a round 19 pin male connector on the other side. We will refer to these connectors and their mates by their brand name: Jaeger.

These cable connections will depend solely on the type of motors connected to your stages. This connection is very sensitive. Please be careful and double check before you make any cable hookup. A wrong connection may destroy the motor driver boards of the MD4.

Determine the type of motors connected to your stages by referring to Figure 1.1. There you will see the possible motors and their corresponding codes names: UE30, UE71, UE72, and UE73.

Look on the back of the MD4. Right below each 19 pin female Jaeger connector you will see a label specifying the type of motor that can be connected to this connector.

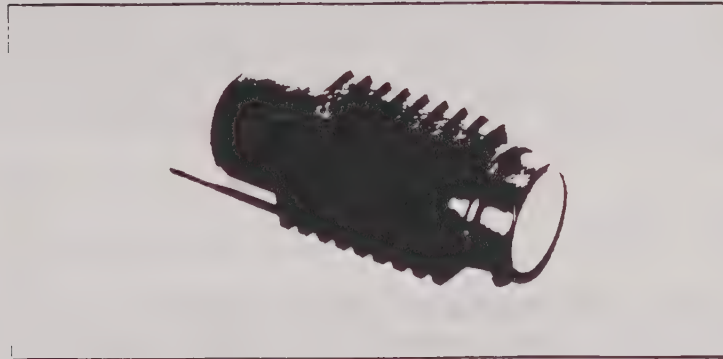
Connect a cable from the 19 pin male Jaeger connector on the stage/motor to the "motor-type matching" 19 pin female Jaeger connector on the back of the MD4.

Repeat the above for the rest of your stages.

1.2 MC4 TO MD4 CABLE CONNECTION

The cable used is approximately 3 feet long and is terminated with a 15 pin male D connector on each end.

Look on the back panels of the MC4 and the MD4. There you will see 15 pin female D connectors labelled W-AXIS, X-AXIS, Y-AXIS, Z-AXIS on the MC4 and INPUT W, INPUT X, INPUT Y, INPUT Z on the MD4.



UE30 Stepping Motor

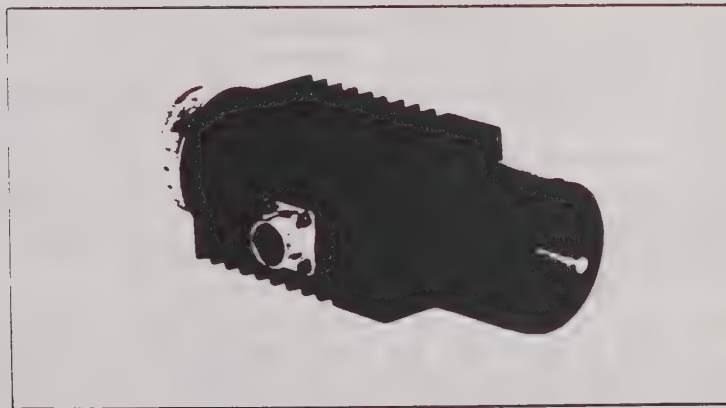
Type: Variable reluctance

Rated voltage: 28 V

Rated current: 200 mA

Resistance/winding: 140-150 Ω

Inductance/winding: 100 mH



UE72 Stepping Motor

Type: Permanent magnet

Rated voltage: 5.3 V

Rated current: 1.6 A

Resistance/winding: 3.3 Ω

Inductance/winding: 9 mH

Figure 1 — UE30 and UE72 stepping motors

Check the stage/motor connected to the OUTPUT W Jaeger connector on the back of the MD4. If you want to reference this stage/motor as your w-axis, connect a cable from the MD4 connector labelled INPUT W to the MC4 connector labelled W-AXIS. If you do not want to reference this stage/motor as your w-axis, then connect a cable from MD4 connector labelled INPUT W to the MC4 connector with the desired axis label. Repeat this procedure for the rest of your stages/motors.

1.3 POWER INPUT

The MC4 and the MD4 will operate properly with either a 110 or 220 VAC line input. Both units are factory preset to operate on 110 VAC. If you need to operate on 220 VAC, please call Klinger Scientific for directions to convert to 220 VAC.

Using the line cords provided, connect each unit to the power line.

1.4 MC4 MOTOR TYPE PROGRAMMING

Once you have connected your motors/stages to the desired axis input/output connectors, you must tell the MC4 what this set-up is for each axis. You will do this by using a two key combination on the MC4 front panel. The two keys are: 1. the reset switch of the axis you want to program and 2. the SELECT switch inside the block of switches designated PROGRAM.

The MC4 stores the motor type information inside its battery back-up RAM. This enables it to remember the motor types during its power-off time. Thus, once you program a motor type for a particular axis, you do not need to re-program on subsequent power ups. On the back of the MC4 there is a switch labelled WRITE PROTECT. One position of the switch allows storing into the battery back-up RAM, and the other position disables this function. When programming motor types, this switch must be set towards dip switch bank S3 to enable storing into the battery back-up RAM. If you try to program motor types with the write protect feature true, the MC4 will beep to indicate an error.

Turn on the MC4 by pressing the POWER switch until it clicks. The displays will all be zeroes, a red bar in the upper left hand display will be illuminating W-AXIS, and the number next to the SELECT switch in the "select window" will be 1. The red bar illuminator identifies the display that corresponds to that axis. The RESET switch on the right of a display is that display's reset switch.

Press and release the AXIS SELECT switch. The red bar will now be illuminating X-AXIS. Repeat this several times and notice which display corresponds to which axis.

You will now tell the MC4 which type of motor is connected to each axis. The procedure is as follows:

1. Place write protect switch towards dip switch bank S3
2. Press and hold down the w-axis reset switch.
3. Press and hold down the program select switch.
4. Release the w-axis reset switch.
5. Release the program select switch

At this time, the number in the select window will change to 3 or a 7 and will flash. The w-axis illuminator bar will also flash to indicate that you are setting the motor type for the w-axis. The specific number that appears will be the code for the motor type that we connected to the w-axis when we set up your system at the factory. The numbers 3 and 7 are codes for the following motor types:

Code	Motor Type
3	UE30
7	UE71, UE72, UE73

If the code number and the type of motor connected to the w-axis match, then go to step 7. Otherwise,

6. Press and release the program select switch.

At this time, the number in the select window will change to the only other possible code number and will flash.

7. Press and release the w-axis reset switch.

At this time, the MC4 will store the motor type information into its battery back up memory. Once set, you do not have to do this again unless you wish to change motor types.

8. Repeat the above procedure for your X,Y, and Z axes by using the X,Y, and Z reset switches, respectively, and the program select switch.
9. Restore write protect switch to its original position

1.5 MOTOR TYPE PROGRAMMING THEORY OF OPERATION

Whenever the above step sequence 1-3 is performed, the number in the select window will show the code for the present motor type programmed for that axis. The number will flash to indicate that you are in this mode of operation. The axis illuminator of the axis you selected for programming the motor type will also flash to indicate which axis is being set.

While in this mode, pressing the SELECT switch will alternate the code numbers between 3 and 7.

To exit this mode of operation and at the same time program the MC4 for the motor type, press and release the reset switch of the corresponding axis.

CHAPTER 2: AXIS CONFIGURATION SWITCH SETTINGS

This chapter describes the switch settings that will tell the MC4 the parameters of your system. For each stage/motor in your system, you will have to determine the following parameters:

1. Is there an origin switch on the stage: Yes/No
2. Is there a top zero pulse option: Yes/No
3. Is there an encoder on the motor: Yes/No
4. If there is an encoder, is it a 48, 200 or a 2000 pulse encoder.

Once you have determined these parameters for each stage, you can then set the MC4 axis configuration switches labelled SW, SX, SY, and SZ. The switch definitions are identical for each axis and represent the following:

<u>Switch Location</u>	<u>Function and Switch Position</u>
1	Origin switch yes - DOWN / Origin switch no - UP
2	Top zero yes - DOWN / Top zero no - UP
3	Encoder yes - UP / Encoder no - DOWN
4	48 or 200 pulse encoder - UP / 2000 pulse encoder - DOWN

NOTE: If switch location #3 was set for no encoder, then switch position of switch location #4 is irrelevant.

NOTE: The 48 pulse encoder is used on UE30 motors. The 200 or 2000 step encoders are used on the UE7x series motors.

Knowing the parameters of the stage connected to the w-axis, set the axis configuration switches labelled SW as follows:

1. If there is an origin switch on the stage, set SW.1 down. If there is no origin switch or if you do not wish to use the origin switch if present, then set SW.1 up.

2. If there is a top zero pulse option, set SW.2 down. If there is no top zero or if you do not wish to use the top zero pulse if present, then set SW.2 up.
3. If there is an encoder, set SW.3 up. If there is no encoder or you do not wish to use the encoder if present, then set SW.3 down.
4. If you set SW.3 for encoder, then set SW.4 up if it is a 48 or 200 pulse per revolution encoder or set SW.4 down if it is a 2000 pulse per revolution encoder.

If you set SW.3 for no encoder, then the position of SW.4 is irrelevant.

Repeat the above steps for the rest of your axes.

CHAPTER 3: LOCAL FRONT PANEL OPERATION

In this chapter you will learn about the use and functions of the MC4 front panel switches. The following features are available via front panel controls of the MC4:

- positioning and moving by single step, low speed and high speed operation
- programming low speed and high speed stepping rates
- automatic home (origin) seeking operation
- selection and execution of internally stored programs
- resetting of front panel displays
- emergency stop of all movement

We will now describe the above mentioned features of the MC4 in greater detail. It is advised that you perform each function as it is described.

Turn on both the MC4 and the MD4 by pressing the POWER switch until it clicks into position.

3.1 FRONT PANEL POSITIONING

Front panel control of movement and positioning is done with the AXIS SELECT switch and the set of switches outlined by the box labelled MANUAL. The manual switches are defined as follows:

- right arrow switch does low speed motion in the positive direction
- left arrow switch does low speed motion in the negative direction
- HIGH SPEED switch does high speed motion when pressed along with a low speed switch

Since there is only one set of manual switches and we can operate up to four axes, we use the AXIS SELECT switch to select which axis the manual switches will control.

Manual positioning consists of the following steps:

1. Press and release the AXIS SELECT switch. A red bar indicator will light to illuminate the axis selected.

Each time you press and release the **AXIS SELECT** switch, the red bar indicator will illuminate the next axis selected.

2. To move a single step at a time on the axis selected, press and release one of the arrow switches.

For UE30 motors, each single step corresponds to a full step of the motor. These motors have 48 steps/revolution; thus one revolution would require 48 single step actuations.

For UE7x motors, each single step corresponds to one-tenth of a full step of the motor. These motors have 200 full steps per revolution; thus one revolution would require 2000 single step actuations.

3. To move continuously at low speed on the axis selected, press and hold down one of the arrow switches. The low speed is factory set at 400 full steps/second. This speed is adjustable. See section 3.2 below.
4. To move continuously at high speed on the axis selected, press and hold down both an arrow switch and the **HIGH SPEED** switch. The high speed is factory set at 2000 full steps/second. This speed is adjustable. See section 3.2 below. The acceleration ramp from low speed to high speed is programmable. See section 7.2.5.

3.2 MANUAL LOW SPEED AND HIGH SPEED PROGRAMMING

Programming the low speed and the high speed for a particular axis involves a two switch combination. One switch is the reset switch of the axis for which you wish to set the speed. The other switch is either arrow switch if you wish to set the low speed or the **HIGH SPEED** switch if you wish to set high speed.

The display of the axis for which you are setting the speed is used to show the speed as you adjust it. As a result, the position display information will be lost.

The MC4 stores the programmed speeds inside its battery back-up RAM. This enables it to remember the set speeds during its power off time. On the back of the MC4 there is a switch labelled **WRITE PROTECT**. One position of the switch allows storing into the battery back-up RAM and the other position disables this function. When programming speeds, this switch must be set towards the dip switch bank S3 to enable storing into the battery back-up RAM. If you try to set a new speed with the write protect feature true, the MC4 will beep to indicate an error.

3.2.1 SETTING LOW SPEED RATE

To set the low speed for the w axis, the steps are as follows:

1. Place write protect switch towards dip switch bank S3

2. Press and hold down the w axis reset switch
3. Press and hold down either arrow switch
4. Release the w axis reset switch
5. Release the arrow switch

At this time, the w axis display will show the present low speed setting. The w axis illuminator will be flashing to indicate that you are setting the w axis speed.

6. To increment the low speed by 10 full steps/second, press and release the right arrow switch. For continuous incrementing, hold down the right arrow switch.
7. To decrement the low speed by 10 full steps/second, perform step 6 with the left arrow switch.
8. To increment/decrement the low speed by 50 full steps/second use the HIGH SPEED switch. First you must establish increment or decrement by pressing the appropriate arrow switch once. Then press the HIGH SPEED switch to change by 50 full steps/second.
9. When you have reached your desired speed, press and release the w axis reset switch to save your low speed setting and to exit this mode.

3.2.2 SETTING HIGH SPEED RATE

To set the high speed for the w axis the steps are as follows:

1. Place write protect switch towards dip switch bank S3
2. Press and hold down the w axis reset switch
3. Press and hold down the HIGH SPEED switch
4. Release the w axis reset switch
5. Release the HIGH SPEED switch

At this time, the w axis display will show the present high speed setting. The w axis illuminator will be flashing to indicate that you are setting the w axis speed.

6. To increment the high speed by 50 full steps/second, press and release the right arrow switch. For continuous incrementing, hold down the right arrow switch.
7. To decrement the high speed by 50 full steps/second, perform step 6 with the left arrow switch.

8. To increment/decrement the high speed by 200 full steps/second use the HIGH SPEED switch. First you must establish increment or decrement by pressing the appropriate arrow switch once. Then press the HIGH SPEED switch to change by 200 full steps/second.
9. When you have reached your desired speed, press and release the w axis reset switch to save your high speed setting and to exit this mode.

Repeat the above procedure described for the w axis for all the other axes by using the RESET switch for that axis and the manual control switches.

When you finish setting speeds, set the write protect switch to its original position.

NOTE: The lowest low speed that may be programmed is 50 full steps per second. The highest high speed is 4000 full steps/second. The smallest gap between the low speed and the high speed allowed is 200 full steps/second. For example, if the low speed is set for 350, the lowest high speed that you may program is 550 full steps/second. You do not have to be wary of these limits. The MC4 will not allow you to exceed them.

NOTE: When you exit this speed setting mode, the display will continue to show the speed you just set. It will not return to the value it had just before you entered this mode.

3.3 ORIGIN SEARCH

The origin search function is an automatic home seeking operation. It returns a stage/motor to a preset position -- a reference point for subsequent positioning. The origin point must be defined and conveyed to the MC4 through the corresponding axis configuration switch on the back of the unit.

The three possible origin points are:

1. The position defined by the location of a switch mounted on a positioning stage
2. The position defined by the location of a switch mounted on a positioning stage in conjunction with a top zero pulse
3. The position that corresponds to the step count of zero on the display for the axis

You will recall from chapter 2 that axis configuration switch locations #1 and #2 (i.e. SW.1, SW.2, SX.1, etc.) are used to set the MC4 for the above origin point definitions. If you set switch location #1 (i.e., SW.1, SX.1, etc.) for no origin switch, the MC4 automatic defaults to the origin point where the display is zero.

Origin point definition number 3 is called a floating origin point. The reason is that you can always define a new position as your origin point by simply moving to a new location and then resetting the display to zero.

To perform origin search, do the following:

1. Select the axis by using the **AXIS SELECT** switch
2. Press and release the **ORIGIN SEARCH** switch

A red indicator light above the **ORIGIN SEARCH** switch will turn on to signal this mode of operation. When the origin search is complete, the indicator light will turn off.

3.4 INTERNAL PROGRAM SELECTION AND EXECUTION

The MC4 has a battery back-up RAM for the purpose of storing many user programs. These programs are entered and edited with a computer. They are stored in the battery back-up RAM during the power off state. Subsequently, a program may be selected and executed without having a host computer present. The **PROGRAM SELECT** and the **PROGRAM START** switches are used for this purpose. The **SELECT** switch allows selection of the first nine programs stored. However, note that more than nine programs may be stored and called for execution. To access programs beyond program nine, a computer must be used. See chapter 7, **GI** command description.

NOTE: When your MC4 was prepared at the factory, it was loaded with five very simple programs. Program 1 moves the w axis 1000 steps in the positive direction at a rate of 2000 steps/second. Then it moves the w axis 1000 steps in the negative direction. Then it loops back to repeat the same process indefinitely. Programs 2, 3, and 4 do the same operation for the X, Y, Z axes, respectively. Program 5 moves all axes simultaneously, does origin search on each axis and toggles output bits.

To execute a program, do the following:

1. Select the program by pressing and releasing the **SELECT** switch
2. Start program execution by pressing and releasing the **START** switch

A red indicator light above the **START** switch will turn on to signal this mode of operation. When the program is done, the indicator light will turn off.

If you wish to stop program execution momentarily, press and release the PAUSE switch. This will cause execution to stop when the present command is done. Note that execution will not stop immediately after you activate pause. The MC4 will pause only after the present command is fully executed. A red indicator light will turn on to signal this mode of operation.

To resume execution after a pause, press and release the PAUSE button again. The red indicator will turn off to signal that pause is now inactive.

3.5 RESETTING FRONT PANEL DISPLAYS

Each axis display has its own reset switch. To perform a reset, simply press and release the RESET switch on the right side of the display you wish to reset.

Resetting a particular axis display does three things:

1. Sets the display readout to zero
2. Sets the present position as the floating origin point
3. Sets the present position as the reference point for the absolute positioning command -- More on this in chapters 6 and 7.

A description of the front panel display and what the numbers represent under different system configurations is presented in chapter 9.

3.6 EMERGENCY STOP

The STOP switch is used to stop all stage movement immediately. Pressing the STOP switch causes the MC4 to stop sending step pulses and returns it to its power up state. When the switch is activated, the MC4 will beep once to acknowledge that it received the stop request.

3.7 REMOTE/LOCAL KEYSWITCH

The purpose and function of this keyswitch will be revealed in the next chapter when we discuss the IEEE-488 interface.

CHAPTER 4: IEEE-488 INTERFACE

4. In addition to the front panel controls for moving and positioning, the MC4 also has a command set which enables you to program it to perform a variety of precise and automated operations. The necessary sequence of commands to perform a certain move must be sent to the MC4 by a host computer or a similar device. To establish a path for the correct transfer of commands a standard interface is necessary between the MC4 and the host computer.

The MC4 supports two standard interfaces. These are the IEEE-488 (also known as GPIB and HP-IB) and the RS-232-C.

The MC4 IEEE-488 interface implements both the listen function and the talk function. The listen function is used to send commands to the MC4. The talk function is used by the MC4 to send status information back to the host computer. Typical status will be:

1. error message if incorrect command sent
2. current display readout
3. command execution in progress or complete
4. end of travel limits true or false
5. listing of stored programs

4.1 IEEE-488 CONNECTOR PIN DEFINITIONS

For connection to the IEEE-488 bus, the MC4 has the standard 24 pin female ribbon connector. Pin assignments for this connector are shown below and are based on the standard IEEE-488 protocol.

IEEE-488 BUS CONNECTOR

E1	DIO1	E 9	IFC	E17	REN
E2	DIO2	E10	SRQ	E18	Gnd. for DAV
E3	DIO3	E11	ATN	E19	Gnd. for NRFD
E4	DIO4	E12	SHIELD	E20	Gnd. for NDAC
E5	EOI	E13	DIO5	E21	Gnd. for IFC
E6	DAV	E14	DIO6	E22	Gnd. for SRQ
E7	NRFD	E15	DIO7	E23	Gnd. for ATN
E8	NDAC	E16	DIO8	E24	LOGIC Gnd.

4.2 MC4 AND THE IEEE-488

We will now describe the steps necessary to establish communications between the MC4 and your host computer. You will test for successful communications by using your host computer to send a few commands to the MC4 and observing the MC4's response. When you are satisfied that the interface is functioning properly, you can move on to chapters 6 and 7 for the description and usage techniques of all MC4 commands. Since your communications interface is set up, you can experiment with each command to fully understand its function.

You will now be asked to do the following:

1. load an IEEE-488 application program into your host computer
2. configure the MC4 dip switches to set communication parameters
3. send commands to the MC4 to verify successful implementation

4.2.1 IEEE-488 APPLICATION PROGRAM

In Chapter 11 we show an application program for several different manufacturers' IEEE-488 interface. If you are using one of these interfaces, enter the program for your interface. If you are using a different manufacturer, study the application program and write a similar one for your interface. The application program does the following:

1. initializes the interface
2. prompts you to enter a command at the keyboard
3. sends the command to the MC4
4. if the command sent was one which requires the MC4 to send back information, the program asks for this information and decodes it

4.2.2 CONFIGURING MC4 FOR IEEE-488

1. Connect a standard IEEE-488 cable from your computer to the IEEE-488 connector on the back of the MC4
2. Set the desired IEEE-488 address for the MC4 by programming dip switch bank S1, accessed from the back of the MC4. S1-1 is the most significant bit and S1-5 is the least significant bit. Our application program is written for the MC4 at address number 6. The dip switch setting for address 6 is:

S1-1	OFF	$0 \times 2^4 = 0$	0
S1-2	OFF	$0 \times 2^3 = 0$	0
S1-3	ON	$1 \times 2^2 = 4$	4
S1-4	ON	$1 \times 2^1 = 2$	2
S1-5	OFF	$0 \times 2^0 = 0$	0

6 ← ADDRESS

- Set the MC4 for the command terminator your computer will use at the end of every command. Switch S3-4 is used for this purpose. The three possible terminators and the S3-4 setting for each is listed below:

Terminator:	S3-4 Setting:
carriage return	ON (UP) ↑
carriage return / line feed	OFF (DOWN) ↓
line feed	OFF (DOWN) ↓

Our application program will output a carriage return/line feed terminator so set S3-4 OFF (or down).

4.2.3 TESTING THE IEEE-488 INTERFACE

The application program is in your computer and the MC4 is configured. You can now send a few commands to the MC4. By observing the response (or lack of one), you can verify if everything is okay or if you will have to spend some time troubleshooting.

- RUN the application program
- At the prompt, enter the command MW and press Return or Enter key

NOTE: Every command string is terminated by a Return or Enter key so we will not keep repeating that statement.

If there is a stage/motor connected to the w axis, (i.e., the + LIMIT and the - LIMIT indicator are not lit), the w axis will move 1000 steps in the positive direction.

NOTE: On power up, the MC4 sets the following defaults:

Number of Steps:	1000
Stepping Rate:	2000 steps/second
Direction:	Positive

NOTE: If the above command does not execute, it may be due to a problem with the IEEE-488 interface. The MC4 is designed to detect some IEEE-488 errors. If it detects an error, it will indicate the error by flashing a number in the program select window and beeping 10 times. Refer to chapter 12 for an explanation and possible solution of the error.

The following set of commands and the expected results will reference only the w axis. After you have done the w axis commands or if you do not have a stage/motor connected to the w axis, repeat the commands for all the other axes by simply replacing the W in each command by the axis (X,Y, or Z) you want to program.

At the prompt, enter the following commands and observe the listed results:

COMMAND: RESULT:

AW	W display reset to zero
NW1234	Set number of steps = 1234
— RW1500	Set high speed step rate = 1500 steps/sec.
MW	W axis moves and display goes to 1234
DW	Read the w-axis display; display on the screen will be: W = +00001234
PW500	W axis moves to position 500
NW2000	Set number of steps = 2000
-W	Set direction to negative
MW	W axis moves and display goes to -1500
OW	W axis moves to origin position; display goes to zero if floating origin or some number if origin defined by a switch
MV	V is an invalid parameter. The MC4 will beep and store the error message into its output buffer.
Q	Branch to the routine in the application program that reads the MC4 output buffer and prints the message to the screen.

The message will be: E02 PARAMETER ERROR

WX

W is an invalid command. The MC4 will beep.

Q

Get and display the error message. The message will be:
E01 COMMAND ERROR

4.3 REMOTE/LOCAL KEYSWITCH

The purpose of this keyswitch is to enable or override the remote enable signal (REN in IEEE-488 terminology) from the controller. The controller will normally be the IEEE-488 interface of your host computer.

When you are using the IEEE-488 interface to operate the MC4, you will probably wish to disable the front panel functions to prevent any accidental switch activations. You can do this by asserting the REN signal of the IEEE-488 true while the MC4 is addressed and turning the MC4 keyswitch to REMOTE.

When you wish to enable the front panel functions, turn the keyswitch to the LOCAL position.

The keyswitch performs its function only after the MC4 IEEE-488 interface senses the REN signal from the controller and it has been addressed. Prior to that time, the front panel functions will be enabled regardless of the position of the keyswitch.

CHAPTER 5: RS-232-C INTERFACE

In addition to the front panel controls for moving and positioning, the MC4 also has a command set which enables you to program it to perform a variety of precise and automated operations. The necessary sequence of commands to perform a certain move must be sent to the MC4 by a host computer or a similar device. To establish a path for the correct transfer of commands a standard interface is necessary between the MC4 and the host computer.

The MC4 supports two standard interfaces. These are the RS-232-C and the IEEE-488 (also known as GPIB and HPIB).

The MC4 RS-232-C interface implements a half-duplex version of the RS-232-C protocol. This means that information can pass from your computer to the MC4 and from the MC4 to your computer, but not simultaneously. The MC4 can only accept or only output information at a given time.

The MC4 accepts and executes commands that are part of its command set. It outputs status information when it deems it necessary and when you ask for it. Typical status will be:

1. Error message if incorrect command sent
2. Current display readout
3. Command execution in progress or complete
4. End of travel limits true or false
5. Listing of stored programs

5.1 RS-232-C CONNECTOR FOR THE MC4

The MC4 uses the standard 25 pin female D connector for coupling to a host computer or data terminal. The MC4 connector is configured as a DCE. Your host computer will be a DTE so you can use straight through cable. See section 5.2.3. The following table lists the MC4 connector pinouts for the RS-232-C.

RS-232-C CONNECTOR

1	Protective Ground	7	Signal Ground
2	Receive Data	8-17	N.C.
3	Transmit Data	18	+12 V
4	N.C.	20	Data Terminal Ready
5	Clear to Send	21-24	N.C.
6	N.C.	25	-12 V

5.2 MC4 AND THE RS-232-C

We will now describe the steps necessary to establish communications between the MC4 and your host computer. You will test for successful communications by using your host computer to send a few commands to the MC4 and observing the MC4's response. When you are satisfied that the interface is functioning properly, you can move on to chapters 6 and 7 for the description and usage techniques of all MC4 commands. Since your communications interface is set up, you can experiment with each command to fully understand its function.

You will now be asked to do the following:

1. Decide whether you will use a hardware handshake or a software handshake to synchronize the data transfers between the MC4 and your computer.
2. Load an RS-232-C application program into your host computer
3. Configure the MC4 dip switches to set communication parameters
4. Send commands to the MC4 to verify successful implementation

5.2.1 HARDWARE/SOFTWARE HANDSHAKE

The MC4 needs a handshake sequence to insure proper reception and execution of all commands. The handshake is such that it prevents you from sending commands when the MC4 is not ready to accept them. The MC4 provides both a hardware and a software handshake. The selection of handshake mode is done with switch S2-4, accessed from the back of the unit.

HARDWARE HANDSHAKE

The hardware handshake uses the following pins on the RS-232-C connector to implement the handshake:

- **Pin 5: Clear to Send**

This signal is an output from the MC4 to your computer

The MC4 asserts clear to send (+12V) when it is ready to accept a new command

- **Pin 20: Data Terminal Ready**

This signal is an input to the MC4 from your computer

The MC4 polls this signal for a true state (+12V) before it sends any data

To send commands to the MC4, simply monitor the state of the clear to send line. Once it is asserted true, send a command. When the MC4 accepts the command, it will assert clear to send false. Continue polling the state of this line for a true signal before sending the next command.

To receive data from the MC4, your computer must set data terminal ready true when it can accept data. If your computer has a communications buffer, you will probably keep this line always true and accept MC4 data when it wants to send it.

SOFTWARE HANDSHAKE

When the MC4 is ready to accept a new command, it outputs an XON character (ASCII DC1, HEX 11). You will receive the XON character immediately when the MC4 is turned on and more importantly, when the MC4 finishes executing the previous command.

Once the MC4 accepts a command and starts to execute it, it will output an XOFF character (ASCII DC3, HEX 13).

The procedure for the software handshake is very simple. Wait to receive an XON character. When received, send a command. Wait to receive an XON character. When received, send another command, etc. As you can see, you do not even have to look for the XOFF character.

When the MC4 is sending data to your computer, it is checking for reception of XON/XOFF characters from your computer. If your communication buffer is large or your program can read data as it is received from the MC4, you will never send XOFF. However, if you cannot accept data at a certain time, you can send XOFF. At that time the MC4 will suspend transmission until it receives a XON character.

NOTE: The MC4 powers up in the XON mode. If you never send XOFF, then you never have to send XON.

5.2.2 RS-232-C APPLICATION PROGRAM

In Chapter 11 we show two application programs for the RS-232-C interface, one using the hardware handshake and one using the software handshake. These programs are written for the IBM-PC and compatibles. You can choose to load one of these programs into your computer or you can use any serial communications program that you may have. The application program does the following:

1. Initializes the interface
2. Prompts you to enter a command at the keyboard
3. Sends the command to the MC4
4. If the command sent was one which requires the MC4 to send back information, the information comes into your computer's communications buffer and is displayed on the screen

5.2.3 CONFIGURING MC4 FOR RS-232-C

You will now configure the MC4 for serial communication by using dip switch bank S2 and S3 on the back of the unit to set the communication parameters. Switch bank S2 is used to set the baud rate and the choice between a hardware or a software handshake (refer to Figure 1). Switch bank S3 is used to set the data word characteristic and the command terminator (refer to Figure 2).

<u>S2-1</u>	<u>S2-2</u>	<u>S2-3</u>	<u>BAUD RATE</u>
ON	ON	ON	110
ON	ON	OFF	150
ON	OFF	ON	300
ON	OFF	OFF	600
OFF	ON	ON	1200
OFF	ON	OFF	2400
OFF	OFF	ON	4800
OFF	OFF	OFF	9600
<u>S2-4</u>		<u>HANDSHAKE</u>	
ON		Software	
OFF		Hardware	

Figure 1: Switch Settings for Baud Rate and Handshake Mode

<u>S3-1</u>	<u>S3-2</u>	<u>S3-3</u>	<u>WORD CHARACTERISTIC</u>
ON	ON	ON	7 bits, even parity, 2 stop bits
ON	ON	OFF	7 bits, odd parity, 2 stop bits
ON	OFF	ON	7 bits, even parity, 1 stop bit
ON	OFF	OFF	7 bits, odd parity, 1 stop bit
OFF	ON	ON	8 bits, 2 stop bits
OFF	ON	OFF	8 bits, 1 stop bit
OFF	OFF	ON	8 bits, even parity, 1 stop bit
OFF	OFF	OFF	8 bits, odd parity, 1 stop bit
<u>S3-4</u>			<u>TERMINATOR</u>
ON			Carriage return
OFF			Carriage return/line feed or line feed only

Figure 2: Switch Settings for Word Characteristics and Terminator

5.2.4 SETTING UP FOR RS-232-C

1. Make an appropriate cable to connect the MC4 and your computer. You will recall that the MC4 is designed to act as a DCE and your computer is most probably configured as a DTE. As a result your cable should look like this:

PIN #:	PIN #:
1 -----	1
2 -----	2
3 -----	3
5 -----	5
7 -----	7
20 -----	20

2. Set the baud rate to 1200 by using the table above. Our application program is written to operate at 1200 baud.
3. Set the word characteristic to match the application program by using the table above. Our application program is set for 7 bits, even parity, and 2 stop bits.
4. Set the handshake mode by using S2-4. Set S2-4 ON (up) for the software handshake and set it OFF (down) for the hardware handshake.

5. If you set up for software handshake mode, and your computer does not provide +12V on pin 20 of its RS-232-C, then you must connect pin 18 and pin 20 of the MC4 RS-232-C connector to supply the +12V necessary.
6. Set the MC4 for the command terminator your computer will use at the end of every command. Switch S3-4 is used for this purpose. Our application program will output only a carriage return terminator thus set S3-4 ON (up).

5.2.5 TESTING THE RS-232-C INTERFACE

The application program is in your computer and the MC4 is configured. You can now send a few commands to the MC4. By observing the response (or lack of one), you can verify if everything is okay or if you will have to spend some time troubleshooting.

1. RUN the application program
2. At the prompt, enter the command MW and press Return or Enter key

NOTE: Every command string is terminated by a Return or Enter key so we will not keep repeating that statement.

If there is a stage/motor connected to the w axis, (i.e. the + LIMIT and the - LIMIT indicator are not lit), the w axis will move 1000 steps in the positive direction.

NOTE: On power up, the MC4 sets the following defaults:

Number of Steps:	1000
Stepping Rate:	2000 steps/second
Direction:	Positive

NOTE: If the above command does not execute, it may be due to a problem with the RS-232-C interface. The MC4 is designed to detect some RS-232-C errors. If it detects an error, it will indicate the error by flashing a number in the program select window and beeping 10 times. Refer to chapter 12 for an explanation and possible solution of the error.

The following set of commands and the expected results will reference only the w axis. After you have done the w axis commands or if you do not have a stage/motor connected to the w axis, repeat the commands for all the other axes by simply replacing the W in each command by the axis (X, Y, or Z) you want to program.

At the prompt, enter the following commands and observe the listed results:

COMMAND:	RESULT:
AW	W display reset to zero
NW1234	Set number of steps = 1234
RW1500	Set high speed step rate = 1500 steps/sec.
MW	W axis moves and display goes to 1234
DW	Read w axis display; display on screen will be: W = +00001234
PW500	W axis moves to position 500
NW2000	Set number of steps = 2000
-W	Set direction to negative
MW	W axis moves and display goes to -1500
OW	W axis moves to origin position; display goes to zero if floating origin or some number if origin defined by a switch
MV	V is an invalid parameter. The MC4 will beep and send an error message to your computer which will be displayed on the screen. The message will be: E02 PARAMETER ERROR
WX	W is an invalid command. The MC4 will beep and the error message will be: E01 COMMAND ERROR

If all the above commands execute as expected, you are ready to move on to Chapters 6 and 7 for a detailed description of all MC4 commands.

CHAPTER 6: AUTOMATED CONTROL/COMMAND SET USAGE

The MC4 has a command set which enables automated and precise control of its operation. The commands may be sent to the MC4 for immediate execution or they may be stored in the MC4's internal battery back up program memory for execution at a later time. The commands are sent to the MC4 using the IEEE-488 interface or the RS-232-C interface.

6.1 SUMMARY OF MC4 COMMANDS

1. Motor Control

- Commands for setting stepping rate

R program stepping rate from 32 to 4000 steps/second

T program step intervals from 2 milliseconds to 100 seconds

- Commands for relative positioning

+/- set direction of movement

N set number of steps for the next move

M start move

- Commands for absolute positioning

A set present position as reference point; also resets front panel display

P move to location specified relative to the defined reference point

- Command for programming acceleration

AC

- Origin search

O

2. Wait for Event

- B** wait for motor to finish moving a predefined number of steps
- UT** wait a predefined number of milliseconds (time delay)
- UE** wait for a high or low condition of an input

3. Status Report/Request

- FS** program status options desired
- ?** report status
- D** read front panel display (i.e. position of motors)

4. Service Request

- FI** program conditions which will generate service request

5. Output Control

- C** set output bit low
- S** set output bit high

6. Stepping and Holding Torque Inhibit/Enable

K

7. Define and Execute Variable Parameters

V

8. Simultaneous and Interdependent Command Execution

<command;command; ... ;command;>

9. Stored Program Editing Commands

- E** enter program entry mode
- QP** end of a single program
- QE** exit program entry mode
- I** insert a program line
- #** delete program lines

L list program lines

\$ comments inside programs

10. Stored Program Execution Command

GI

11. Jump Commands

H define a "jump to" label

J jump to the specified label

6.2 THEORY AND DESCRIPTION OF COMMANDS

An MC4 command is always composed of a sequence of ASCII characters terminated by a carriage return, a carriage return/line feed combination, or a line feed.

- the first character is the command letter of the function to be performed
- for commands associated with an axis, the second character represents the desired axis (W,X,Y,Z or A for all axes)
- subsequent characters are numerical parameters for that command

Examples:

RX1000 ;set the high speed rate for the X axis at 1000 steps/sec.

NY5000 ;set the number of steps for the next Y axis move to 5000

+Y ;set Y axis move direction to positive

MY ;start Y axis move

S3 ;set output bit #3 high

OZ ;perform origin search on Z axis

DA ;display all axes positions

FS1F ;program status response desired

6.2.1 PRINCIPLES OF COMMAND EXECUTION

The MC4 has three different modes of command execution:

1. Sequential

A command is accepted and executed when the previous command is done. If you try to send a command while the MC4 is busy executing the previous command, your interface will go into a looping mode waiting for the MC4 to become ready. At that time, the command will be sent and the MC4 will proceed to execute it.

Examples:

MX ;move X

MY ;move Y when X move complete

2. Simultaneous

The execution of all commands starts at the same time. To designate this mode, the set of commands must be enclosed in arrowhead brackets and each command must be followed by a semicolon, including the last one.

NOTE: A bunch of simultaneous commands will necessarily be sent in the sequential mode of operation.

Examples:

<MW;MX;> ;start moving W and X at the same time

<MX;S2;C2;> ;move X and at the instant X begins to move, pulse output bit #2

3. Interdependent

The start of execution of a command is dependent on the extent of execution of another command.

Examples:

NX1000 ;set X axis for 1000 steps
<MX;BX500;MY;> ;move X; when X outputs 500 steps, move Y
<MY;UT200;MW;> ;move Y; 2 seconds after Y move started, move W

6.2.2 SIMULTANEOUS AND INTERDEPENDENT MODE COMMANDS

The following subset of the MC4 command set can be used in these modes of operation. Using a command not specified below will be considered an error. The MC4 will beep and send an appropriate error message.

B	;wait for specified number of steps to be completed
C	;clear output bit
H	;declare jump label
J	;jump to specified label
M	;start move on designated axis
S	;set output bit high
UE	;wait for external input level
UT	;wait for specified time delay
+	;set direction for relative move to plus
-	;set direction for relative move to minus
V	;define and use variable parameter

6.3 COMMAND PROGRAMMING MODES

The MC4 has three programming modes:

1. real-time mode
2. program entry and edit mode
3. program execution mode

6.3.1 REAL-TIME PROGRAMMING MODE

The MC4 goes automatically into this mode when:

- it is turned on
- when you exit program entry and edit mode (i.e. issue a QE command)
- and when a stored program is finished executing

In this mode, commands are executed as they are received. You have to wait until the previous command is done before sending another command. The implementation of this timing requirement is dependent on the communication interface you are using (IEEE-488 or RS-232-C).

IEEE-488

When the MC4 accepts a command and starts to execute it, it asserts the "not ready for data" signal on the IEEE-488 interface. Your computer interface senses this signal and does not allow you to send any further commands. Once the previous command is done, the MC4 will assert "ready for data" and now your computer interface will allow you to send another command.

RS-232-C

In the hardware handshake mode, the MC4 will assert "clear to send" false to the interface when it accepts a command and starts to execute it. Your computer interface should be set up to check the status of this line before sending a command. In this case, sensing a "clear to send" false signal, it should wait for this line to go to "clear to send" true. When the previous command is done, the MC4 will assert "clear to send" true.

In the software handshake mode, the MC4 sends an XON character (ASCII 11) to your computer when it is ready to accept another command. Therefore, you must wait to receive the XON before sending another command.

6.3.2 PROGRAM ENTRY AND EDIT MODE

In this mode, a sequence of commands is sent and stored in a program buffer for later execution. The program buffer is battery powered during the MC4 power-off state. This enables all entered programs to be saved and available for execution at any time. Once programs have been entered, they can be read from the program buffer and modified easily.

In this mode, as in real-time mode, a handshake is necessary to transfer the commands. Here however, the MC4 will signal "not ready" only momentarily because it only converts the command to a coded, compact form and stores it in memory. It does not execute it.

6.3.3 PROGRAM STORAGE ORGANIZATION

The capacity of the program buffer is 8 kilobytes. The commands are stored in a coded, compact form. This enables more commands to be stored than if they were stored as ASCII characters. For example, a command like NX1234567 is stored in 4 bytes rather than the 9 bytes that would be required if each character's ASCII code were stored.

The stored programs are organized in terms of a program line. The concept of the program line is essential to the correct execution of all program entry and edit commands. A program line can be:

1. a single command, a single simultaneous command, or a single interdependent command

2. a group of commands, a group of simultaneous commands, or a group of interdependent commands

The MC4 keeps track of all program lines, numbering them in ascending order starting from zero, from the next available line, or from the line number specified with the E command. The following commands all require the proper use of program line numbers.

- enter program entry mode: Ennnn
- list program: Lnnnn
- insert a program line: Innnn
- delete program lines: #nnnn
- execute from program line: GInnnn

nnnn = line number

You enter program entry mode by sending the E command with a starting line number. After this, you do not have to send program line numbers. You just send commands and the MC4 automatically assigns line numbers to your commands.

The MC4 allows you to store many different programs. Each program is delineated by the command QP. When you have entered all the commands for a particular program, you must issue a QP command to tell the MC4 that this is the end of this program.

If you wish to start entering another program, simply continue issuing commands and when done, end the program with the QP command. If you wish to exit program entry mode, send the QE command.

This will return you to real-time mode. Now you may wish to have the MC4 execute one of your stored programs. This can be the program you just entered or any other program previously stored. If you do not wish to execute any programs, the MC4 is ready to execute any other commands in its command set.

6.3.4 PROGRAM EXECUTION MODE

In this mode, the MC4 will execute a program previously stored in its program memory. You can enter this mode in one of two ways. One is with the front panel switches associated with program selection and execution. The other is with your computer, by issuing a GInnnn command. nnnn specifies the line number at which you wish to start executing your program. The MC4 will continue to execute commands until it encounters a QP command. At that time, it will exit program execution mode and re-enter real-time mode.

NOTE: There is a difference between executing programs with the front panel controls and executing them with the GI command. With the front panel controls, execution starts with the first command of the program selected. However, with the GI command you can specify a line number and therefore start execution anywhere you want.

6.3.5 PRACTICE COMMAND PROGRAMMING MODES

If you followed the instructions in Chapter 4 or Chapter 5 you presently have a functioning interface between the MC4 and your computer, and you have been sending commands to the MC4 in the real-time programming mode. We will now present commands that illustrate the principles of the other two programming modes of the MC4: program entry and program execution.

Even if your interface is not set up, please go through the following section because it is very informative.

First command: L

The MC4 will respond by sending a listing of the programs stored in its memory. These programs were loaded into the MC4 before it was shipped to you. Note the line number of the command QE, which is the last command. You will use this line number to enter program entry mode at the correct location. For the purpose of illustration, this exercise will assume that this line number is 60. However, wherever you see line number 60 below, you must use the line number of command QE observed above.

Second command: E60

NOTE: If you do not have a stage/motor connected to an axis called out below, please substitute the axis designator of the ones you do have.

Subsequent commands:

OX
OY
NX1000
+X
MX
NY2000
-Y

```

MY
QP
OX;OY:NX1000;+X;MX;NY2000;+Y;MY
QP
OW
OX
OY
NW300
NX100
NY200
-W
+X
-Y
<MW;MX;>
<MW;MY;>
QP
OW;OX;OY:NW300;NX100;NY200;-W;+X;-Y;<MW;MX;>;<MW;MY;>
QP
QE

```

After you have entered these commands, you may wish to get a listing from the MC4. If so, enter L60. (Remember 60 = line number parameter used with the E command above; the second command issued). The MC4 will send a listing of the command statements in its memory starting from the location specified with the L command.

The listing will look as follows except the line numbers may be different.

```

0060 OX
0061 OY
0062 NX1000
0063 +X
0064 MX
0065 NY2000
0066 -Y
0067 MY
0068 QP
0069 OX;OY:NX1000;+X;MX;NY2000;-Y;MY
0070 QP
0071 OW
0072 OX
0073 OY
0074 NW300
0075 NX100

```

```
0076 NX100
0077 -W
0078 +X
0079 -Y
0080 <MW;MX;>
0081 <MW;MY;>
0082 QP
0083 NW300;NX100;NY200;-W;+X;-Y;<MW;MX;>;<MW;MY;>
0084 QP
0085 QE
```

Let us now execute these programs and see what happens.

Command: GI60

Response: Origin search on X axis followed by origin search on Y axis. Move X 1000 steps in the positive direction. Move Y 2000 steps in the negative direction

Command: GI69

Response: Same as above.

Execute the last two programs by issuing the commands: GI71 and then GI83. You will notice that these two programs also have identical responses. As you can see, the same task can be accomplished by different program line structuring.

The motivation for placing more than one command on a single line is that you can tie together all logically connected commands which may make readability and understanding quicker. Also, the MC4 allows only 9999 line numbers. If you are going to have many programs, you may run out of line numbers.

The disadvantage of placing too many commands on a single line is that if you must edit a single command in that line, you must re-type the whole line. Also you cannot insert or delete a command inside a program line consisting of several commands.

6.4 STATUS REPORT/REQUEST

The following status information is available from the MC4:

1. Descriptive error message if command sent is invalid
2. Descriptive error message if error occurs during execution
3. Axis end of travel limits true or false
4. Front panel display reading while stationary and "on the fly"
5. Axis moving or stationary
6. Logic level of 3 input bits

The above statuses are available while the MC4 is idle, and more importantly, statuses 2-5 are available when it is executing positioning commands.

Statuses 1 and 2 above will be available only when the error condition occurs. Status #3 is available when an end of travel limit occurs and is considered an error. Status #3 is also available whenever you query the MC4 for its status.

When an error does occur, the MC4 will beep twice, very quickly, and store the descriptive error message into its output buffer. If your interface is IEEE-488, you must then issue a read statement (the exact command will depend on your IEEE-488 interface manufacturer) to the interface which will place the MC4 in the Talker mode. The MC4 will then send the contents of its output buffer to your computer.

If your interface is RS-232-C, the MC4 will automatically send the error message to your computer without you asking for it. However, this will occur only if:

- for the hardware handshake mode, your data terminal ready line is true
- for the software handshake mode, the MC4 is in the XON state. The MC4 will be in the XON state if you have not sent an XOFF or if you have sent an XOFF, you subsequently sent an XON.

Statuses 3-6 are available whenever you wish to know them. This can be while the MC4 is idle, waiting for a command, or executing a positioning command or a wait command.

The steps necessary to have the MC4 report its status at any time you want are:

1. When the MC4 is idle, you must tell it which status options you want. You do this with the FS command. This command is explained below.
2. At the instant you want a status report, you do the following based on your interface and the processing state of the MC4:

IEEE-488

MC4 Idle:

- first send the ? command
- then issue a read statement to the interface to place the MC4 in the Talker mode

MC4 Processing Command:

- just issue a read statement to the interface to place the MC4 in the Talker mode. The MC4 assumes you want status information.

MC4 Idle or Processing Command:

- send the ? command and the MC4 will automatically send the status report.

6.4.1 STATUS FORMAT COMMAND

Before you can ask the MC4 its status, you must tell it what particular status you want. You do this by using the FS (format status) command. To the FS command you append a hexadecimal byte which represents the status information you want. If you want a particular status, you set that status' bit location to 1; otherwise you set it to 0. The following table shows the bit locations within the byte assigned to each status.

Hexadecimal Byte: B7 B6 B5 B4 B3 B2 B1 B0

- | | |
|--------|---|
| B0 = 1 | Report axis moving/stationary status |
| B0 = 0 | Do not report axis moving/stationary status |
| | |
| B1 = 1 | Report W and X axis end of travel limits true/false status |
| B1 = 0 | Do not report W and X axis end of travel limits true/false status |
| | |
| B2 = 1 | Report Y and Z axis end of travel limits true/false status |
| B2 = 0 | Do not report Y and Z axis end of travel limits true/false status |
| | |
| B3 = 1 | Report Logic level of 3 input bits |
| B3 = 0 | Do not report Logic level of 3 input bits |
| | |
| B4 = 1 | Report W axis display readout |
| B4 = 0 | Do not report W axis display readout |
| | |
| B5 = 1 | Report X axis display readout |
| B5 = 0 | Do not report X axis display readout |
| | |
| B6 = 1 | Report Y axis display readout |
| B6 = 0 | Do not report Y axis display readout |
| | |
| B7 = 1 | Report Z axis display readout |
| B7 = 0 | Do not report Z axis display readout |

Examples:

1. We want axis moving/stationary status, the W axis display and the Y axis display.

HEX byte: 0101 0001 ---> 51H

MC4 Command: FS51

2. We want axis moving/stationary status, W and X end of travel limits, the W axis display, and the X axis display.

HEX byte: 0011 0011 ---> 33H

MC4 Command: FS33

3. We want all statuses

HEX byte: 1111 1111 ---> FFH

MC4 Command: FSFF

6.4.2 MC4 STATUS RESPONSE FORMAT

The MC4 status response is a string of coded bytes. Your application program will read this string into a string variable. It will then have to manipulate the contents of the string variable to extract the information contained within it.

To help you distinguish which status a particular status byte represents, each status byte is preceded by a header byte to identify it. Following the header byte is the status byte. The process for decoding the status string is to search the string for the header and then read the next byte as the status byte.

The form of the MC4 status report string is:

header1status1header2status2...headerNstatusN carriage return/line feed

As you can see, the header and status bytes are all bunched together and terminated by a carriage return/line feed.

■ Moving/Stationary Status Byte

Header Bit Pattern: 0011 0000 ---> ASCII 0

Status Bit Pattern: 0100 B3 B2 B1 B0

Status Bit Interpretation:

B0 = 1 W axis moving	B2 = 1 Y axis moving
B0 = 0 W axis stationary	B2 = 0 Y axis stationary
B1 = 1 X axis moving	B3 = 1 Z axis moving
B1 = 0 X axis stationary	B3 = 0 Z axis stationary

Example: You desire moving/stationary status (FS01) and the W axis is moving when you ask for status. The MC4 response will be the ASCII string 0ACR/LF; CR = carriage return; LF = line feed

■ W and X axis End of Travel Limits

Header Bit Pattern: 0011 0001 ---> ASCII 1

Status Bit Pattern: 0100 B3 B2 B1 B0

Status Bit Interpretation:

B0 = 1 W axis plus end of travel limit true
B0 = 0 W axis plus end of travel limit false
B1 = 1 W axis minus end of travel limit true
B1 = 0 W axis minus end of travel limit false
B2 = 1 X axis plus end of travel limit true
B2 = 0 X axis plus end of travel limit false
B3 = 1 X axis minus end of travel limit true
B3 = 0 X axis minus end of travel limit false

Example: In addition to moving/stationary status, assume you desire W and X axis end of travel limit (FS03). At the time you ask for status, W axis is moving and the X axis plus end of travel limit is true. The MC4 response will be the ASCII string: 0A1DCR/LF

■ Y and Z axis End of Travel Limits

Header Bit Pattern: 0011 0010 ---> ASCII 2

Status Bit Pattern: 0100 B3 B2 B1 B0

Status Bit Interpretation:

B0 = 1 Y axis plus end of travel limit true
B0 = 0 Y axis plus end of travel limit false

B1 = 1 Y axis minus end of travel limit true
B1 = 0 Y axis minus end of travel limit false

B2 = 1 Z axis plus end of travel limit true
B2 = 0 Z axis plus end of travel limit false

B3 = 1 Z axis minus end of travel limit true
B3 = 0 Z axis minus end of travel limit false

Example: Building on the above status response, you also desire the Y and Z axis end of travel limits status (FS07). Assume the two previous statuses are the same and in addition, the Y axis minus end of travel limit is true. The MC4 response will be the ASCII string: 0A1D2BCR/LF

■ 3 Bit Input Status

Header Bit Pattern: 0011 0011 ---> ASCII 3

Status Bit Pattern: 0100 B3 B2 B1 B0

Status Bit Interpretation:

B0 = 1 Input bit 1 high
B0 = 0 Input bit 1 low

B1 = 1 Input bit 2 high
B1 = 0 Input bit 2 low

B2 = 1 Input bit 3 high
B2 = 0 Input bit 3 low

B3 = 1 No information
B3 = 0 No information

Example: In addition to all the above, you also desire 3 bit status (FS0F). Assume the previous three statuses are the same, and in addition, input bit 2 is low when you ask for status. The MC4 response will be the ASCII string: 0A1D2B3ECR/LF.

■ Displays Status

The display information is sent as ASCII characters, so your program should interpret it as such. Each displays numerical information is preceded by two ASCII characters that identify it. These are: W =, X =, Y =, and Z =.

Header Bit Pattern:

W axis display 0101 0111 ---> ASCII W
X axis display 0101 1000 ---> ASCII X

Y axis display	0101 1001	---> ASCII Y
Z axis display	0101 1010	---> ASCII Z

Status Bit Interpretation:

In this case, the individual bits following the header have no specific meaning. However, the bytes formed by these bits should be interpreted as numerical ASCII characters.

Example: In addition to all the above, you also desire the X and Y display information (FS6F). Assume all previous statuses are the same and, in addition, w-axis display is at +01234567 and the X axis display is at -07654321. The MC4 response will be the ASCII string:

0A1D2B3EW = +01234567 X=-07654321 CR/LF

■ Wait Status

This status is enabled all the time. It cannot be selected or de-selected with a bit in the FS command. If the MC4 is in a wait state at the time you request a status report, a byte in the status response string will indicate this state. If the MC4 is not in a wait state, then only the programmed status bytes will be sent.

Header Bit Pattern: 0011 0100 --> ASCII 4

Status Bit Pattern: 0100 B3 B2 B1 B0

Bit Interpretation:

B0 = 1 Executing B command: wait for number of steps to be completed
 B0 = 0 Not executing B command

B1 = 1 Executing UE command: wait for external input level change
 B1 = 0 Not executing UE command

B2 = 1 Executing UT command: wait for time out
 B2 = 0 Not executing UT command

B3 = 1 No information
 B3 = 0 No information

Example: Assume all above statuses are true and the MC4 is executing the UT command when you ask for status. The MC4 response will be the ASCII string:

4D0A1D2B3EW = +01234567 X=-07654321 CR/LF

NOTE: If the wait status is true when you ask for status, that information will be in the first two bytes of the status response string.

NOTE: If there is a status you do not desire, you inform the MC4 of this fact with the FS command. The resulting status report string from the MC4 will simply have the undesired status byte missing from the sequence. For example: You only desire moving/stationary, 3 bit input, and X axis and Y axis display status, and the MC4 is not in the wait state. The MC4 status response will be the ASCII string:

0A3EW = +01234567 X = -07654321 CR/LF

6.4.3 STATUS REPORT PRACTICE

If you followed the instructions in Chapter 4 or Chapter 5 and presently have a functioning interface between the MC4 and your computer, you are set to experiment with the status reporting feature of the MC4. If you study the practice application program in Chapter 11 (and currently in your computer) you will notice that starting at line number 900 or line 1000 for RS-232-C hardware handshake, is the software that decodes the status report string of the MC4. This is done by searching the bytes of the string for a header byte. Once found, the program branches to the routine that then decodes the information contained in the next byte. If the header indicates display information, then we simply treat the subsequent bytes as ASCII characters.

■ IEEE-488

First command: FSF1 ;we want moving/stationary and all display status

Second command: ? ;MC4 response is that all axes are stationary and the
;display readout

Subsequent commands:

NW5000

NX3000

RA500

MW ;start moving W

While W is moving, enter Q <RETURN> at your keyboard. (In this case, Q is not an MC4 command. It is used as a flag to cause the application program to branch to the software which places the MC4 in the Talker mode.)

MC4 response: W axis moving, all others stationary
 W axis display readout as it is changing, all others no change

While W is moving and when it stops, enter Q <RETURN> many times.

MC4 response: Same as above, except the W display readout is changing.

When W stops, the MC4 will report that condition along with the final readout of the W display.

Next command: <MW;MX;> ;start moving W and X simultaneously; X will
;stop before W

Enter Q <RETURN> many times

MC4 response: At first it will indicate W and X moving, Y and Z stationary and the display readouts at the time you entered Q <RETURN>.

Then when X stops, it will indicate W moving, X,Y,Z stationary and the display readouts at the time you entered Q <RETURN>.

Finally when W stops, it will indicate W,X,Y,Z stationary and the final display readouts.

■ RS-232-C

The practice commands and sequence are the same as above for the IEEE-488, except, wherever it says to enter Q <RETURN>, enter ? <RETURN> instead.

For RS-232-C, you must send the status request command (? command) when you want a status report.

You may wish to experiment further by sending different parameters with the FS command and observing the MC4 status response.

6.4.4 ERROR CHECKING AND REPORTING

The MC4 will only execute valid commands and valid combinations of commands. A valid command is one that conforms to the specifications listed in chapter 7. If you send an invalid command, the MC4 will beep twice, very quickly, and store an error message in its output buffer. If your interface is IEEE-488, you should execute a read statement that will place the MC4 in the Talker mode. If your interface is RS-232-C, the MC4 will automatically send the error message. The format of the error message will be:

Enn ERROR MESSAGE Carriage Return/Line Feed

E is the header byte telling you an error message is coming

nn is the code number for the type of error.

ERROR MESSAGE is a description of the error

See chapter 12 for the listing of all possible errors and their code numbers.

Typical errors may include:

- numerical parameter too small or too large
- not specifying axis designator (W,X,Y,Z, or A)
- sending program entry and edit commands while in program entry mode
- sending an invalid command in the simultaneous and interdependent mode of execution. Only a subset of MC4 commands are allowed in these modes. Refer to section 6.5.4 for a listing of commands that may be used in these modes.

Examples:

Command: RW4500

MC4 response: Beep,beep: E02 PARAMETER ERROR

Command: WX

MC4 response: Beep,beep: E01 COMMAND ERROR

Command: <MW;RY2000;MY;>

MC4 response: Beep,beep: E01 COMMAND ERROR
(R is not allowed in simultaneous mode)

6.5 SERVICE REQUEST GENERATION

Service requests are only available for the IEEE-488 interface. It is part of the protocol.

The MC4 may be programmed to generate a service request signal when any of the following five events occur. These events are:

1. when the W axis completes its move
2. when the X axis completes its move
3. when the Y axis completes its move
4. when the Z axis completes its move
5. when any axis reaches a travel limit

MC4 commands M, P, and O are commands that initiate a move. Thus at the completion of any of these commands for any of the four axes, the MC4 can generate a service request.

The steps necessary to enable the MC4 to generate a service request and subsequently send a status byte to your computer for interpretation are:

1. When the MC4 is idle, you must tell it which of the five events listed above, you want to generate a service request. You will do this with the FI command. This command is explained below.
2. When a selected event occurs, the MC4 will generate a service request signal on the IEEE-488 interface. In response to this signal, your computer will output a serial poll command to the MC4.
3. In response to the serial poll command, the MC4 will send a status byte to your computer. The information represented by each bit of the status byte is described below.

6.5.1 SERVICE REQUEST ENABLE COMMAND

When any of the events occur that can generate a service request, the MC4 will check if it should output a service request signal. If this particular event is enabled, the MC4 will output a service request signal, otherwise it will not. You will use the FI command to enable and disable particular events to generate service requests. To the FI command, you will append a hexadecimal byte which represent enabled and disabled events. If you want a particular event to generate a service request, you set that event's bit location to 1, otherwise you set it to 0. The following table shows the bit locations assigned to each event.

Hexadecimal Byte: B7 B6 B5 B4 B3 B2 B1 B0

- B0 = 1 Generate service request when W axis move is completed
B0 = 0 Do not generate service request when W axis move is completed
- B1 = 1 Generate service request when X axis move is completed
B1 = 0 Do not generate service request when X axis move is completed
- B2 = 1 Generate service request when Y axis move is completed
B2 = 0 Do not generate service request when Y axis move is completed
- B3 = 1 Generate service request when Z axis move is completed
B3 = 0 Do not generate service request when Z axis move is completed
- B4 = 1 Generate service request when any axis reaches a travel limit
B4 = 0 Do not generate service request when any axis reaches a travel limit
- B5 - B7 Not presently assigned

CHAPTER 7: DETAILED DESCRIPTION OF MC4 COMMANDS

In this chapter we will first show a summary of MC4 commands grouped according to associated function. Then we present each command in alphabetical order and list everything you should know about it.

7.5.1 SUMMARY OF MC4 COMMANDS

1. Motor Control

- Commands for setting stepping rate

R program stepping rate from 32 to 4000 steps/second

T program step intervals from 2 milliseconds to 100 seconds

- Commands for relative positioning

+/- set direction of movement

N set number of steps for the next move

M start move

- Commands for absolute positioning

A set present position as reference point; also resets front panel display

P move to location specified to the defined reference point

- Command for programming acceleration

AC

- Origin search

O

2. Wait for Event

B wait for motor to finish moving a predefined number of steps

UT wait a predefined number of milliseconds (time delay)

UE wait for a high or low condition of a input

3. Status Report/Request

FS program status options desired

? report status

D read front panel display (i.e. position of motors)

4. Service Request

FI program conditions which will generate service requests

5. Output Control

C set output bit low

S set output bit high

6. Stepping and Holding Torque Inhibit/Enable

K

7. Define and Execute Variable Parameters

V

8. Simultaneous and Interdependent Command Execution

<command;command; ... ;command;>

9. Stored Program Editing Commands

E enter program entry mode

QP end of a single program

QE exit program entry mode

I insert a program line

delete program lines

L list program lines

\$ comments inside programs

10. Stored Program Execution Command

GI

11. Jump Commands

H define a "jump to" label

J jump to the specified label

7.5.2 ALPHABETICAL LISTING AND DESCRIPTION OF COMMANDS

COMMAND: A (W, X, Y, Z, A)

FUNCTION:

1. Reset display counter(s) to zero
2. Set present position as the floating origin point
3. Set present position as the reference point for the absolute positioning command (i.e., P command)

EXAMPLE: AX ; perform functions 1-3 on the X axis
AA ; perform functions 1-3 on all axes

COMMAND: AC (W, X, Y, Z, A)n.n

FUNCTION: Program acceleration ramp from low speed to high speed.

RANGE: n.n = 0.1 second - 2 seconds

COMMENTS: The acceleration command parameter represents the time the MC4 will accelerate from 50 Hz to 4000 Hz. Thus if you program a stepping rate lower than 4000 Hz that stepping rate will be reached faster than the time specified in the AC command. As you vary the time to ramp to 4000 Hz you also vary the time to ramp to all lower stepping rates.

The acceleration ramp is always exponential. This means that it is steep at the beginning and becomes asymptotic as it approaches 4000 Hz. For example, if you program ACA2, and your desired high speed rate is 2000 Hz, the MC4 will ramp to 2000 Hz in less than 1 second.

Each axis can be programmed for its own acceleration ramp.

The programmed acceleration is valid for all modes of motion: program, command and manual modes of operation.

This command can not be executed in simultaneous execution mode. i.e., <MW;MX;AC1.5;MY;MZ;>

The AC command parameter is preserved during power-down. Consequently, the desired acceleration does not have to be reissued after power-up. (Default is ACA.4)

EXAMPLE: ACW.1 ;ramp to 4000 Hz in 0.1 seconds on the W axis
ACA1.7 ;ramp to 4000 Hz in 1.7 seconds on all axes

COMMAND: B (W, X, Y, Z) nnnnnnn

FUNCTION: Wait for the specified axis to complete nnnnnnn steps and then start executing the next command

RANGE: nnnnnnn = 1 to 7999999

COMMENTS: This command allows you to coordinate the movement of several axes. It also can be used, in conjunction with the C and S command, to signal the execution of n number of steps along the whole movement.

EXAMPLE: <MW;BW500;MX;>
• start W, when W reaches 500 full steps, start X

<MW;BW100;S1;C1;BW200;S1;C1;BW300;S1;C1;>
• start W, when W reaches 100 full steps, pulse output #1, when W reaches 200 full steps, pulse output #1, etc.

COMMAND: C (1, 2, 3, 4, 5, 6, 7, 8)

FUNCTION: Set specified output bit number low.

RANGE: Output bits are numbered from 1 to 8

COMMENTS: The output bits are available on the MC4 General Purpose Input/Output (GPIO) connector. Refer to chapter 8 for the pin-outs. These outputs are open-collector, so you must place a pull-up resistor (value: 1K ohm to 10K ohm) between a power supply and the output pin. For the power supply, you may use the +5 volts or the +12 volts available on the GPIO connector.

EXAMPLE: C1 ; set output number 1 low

<C6;C7;C8;> ; set outputs number 6,7, and 8 low

COMMAND: D (W, X, Y, Z, A)

FUNCTION: Read front panel display and store data into output buffer

COMMENTS: For IEEE-488 interface, you must issue a read statement to the interface to place the MC4 into the Talker mode. At that time the MC4 will send the contents of its output buffer to your computer.

For RS-232-C interface, the MC4 will automatically send the display data to your computer in response to the D command.

The format of the display data sent by the MC4 is:

- in response to DW: W = ±xxxxxxxx
- in response to DY: Y = ±xxxxxxxx
- in response to DA: W = ±xxxxxxxx X = ±xxxxxxxx
Y = ±xxxxxxxx Z = ±xxxxxxxx

All characters sent are ASCII coded, including the 5 spaces between each display readout.

The response string is always terminated by a carriage return/line feed combination regardless of the setting of switch S3-4, which only tells the MC4 how its commands are terminated.

EXAMPLE: DX ;read X axis display
DA ;read all axes displays

COMMAND: Ennnn

FUNCTION: Enter program entry mode at line number nnnn

RANGE: nnnn = 0 to 9999

COMMENTS: Line numbers less than nnnn remain unchanged

Line numbers greater than or equal to nnnn are destroyed.

If nnnn is greater than the number of existing program lines, then command storage will begin at the last program line. The last program line will be QE command line number. This will be overwritten with the first command of the new program.

If `nnnn` is not specified, `nnnn` defaults to 0000. Command storage will start at line number 0000. This will erase all previously stored programs.

You should normally request a listing of your existing programs before issuing the E command. By sending the L command, the MC4 will send a listing of existing programs to your computer. The listing consists of line numbers and commands. By seeing the position of your other programs, you will be able to specify the correct line number for the E command.

CAUTION: If you accidentally enter a wrong line number or forget one, you may be in danger of erasing good programs.

DO NOT enter QE, hoping to exit and try again. Instead, press the STOP switch on the MC4 front panel. This will take you out of program entry mode safely and you may then enter the E command with the proper line number.

EXAMPLE: E ;enter program entry mode at line number 0000
 E20 ;enter program entry mode at line number 0020

COMMAND: FShh

FUNCTION: Program the MC4 for status options desired

RANGE: hh = 00H to FFH

COMMENTS: The parameter hh is binary data where the value (1 or 0) of each bit represents your desire or non-desire of the particular status that it represents.

hh byte = B7 B6 B5 B4 B3 B2 B1 B0

B0 = 1 want axis moving/stationary status

B0 = 0 do not want axis moving/stationary status

B1 = 1 want W and X end of travel limits status

B1 = 0 do not want W and X end of travel limits status

B2 = 1 want Y and Z end of travel limits status

B2 = 0 do not want Y and Z end of travel limits status

B3 = 1 want 3 bit input status (E1, E2, and E3 on GPIO)

B3 = 0 do not want 3 bit input status

B4 = 1 want W axis display readout

B4 = 0 do not want W axis display readout

B5 = 1 want X axis display readout
 B5 = 0 do not want X axis display readout

 B6 = 1 want Y axis display readout
 B6 = 0 do not want Y axis display readout

 B7 = 1 want Z axis display readout
 B7 = 0 do not want Z axis display readout

You must send this command while the MC4 is idle. This is because the MC4 will not accept commands while it is busy executing another command. However, while executing a command, the MC4 will respond to a status request command.

EXAMPLE: FSF1 ;we want axis moving/stopped status and all displays

 FS66 ;we want all axis end of travel limits status and the X and Y displays

COMMAND: FIhh

FUNCTION: Program/select events that will generate service request signal

RANGE: hh = 00H to FFH

COMMENTS: The parameter hh is binary data where the value (1 or 0) of each bit represents your desire or non-desire to have a particular event to generate a service request.

hh byte: B7 B6 B5 B4 B3 B2 B1 B0

B0 = 1 Generate service request when W axis move is completed
 B0 = 0 Do not generate service request when W axis move is completed

B1 = 1 Generate service request when X axis move is completed
 B1 = 0 Do not generate service request when X axis move is completed

B2 = 1 Generate service request when Y axis move is completed
 B2 = 0 Do not generate service request when Y axis move is completed

B3 = 1 Generate service request when Z axis move is completed
 B3 = 0 Do not generate service request when Z axis move is completed

B4 = 1 Generate service request when any axis reaches a travel limit
 B4 = 0 Do not generate service request when any axis reaches a travel limit

B5 - B7 Not presently assigned

EXAMPLE: We want to generate service requests whenever the W axis move is completed, the Y axis move is completed, or an axis reaches a travel limit.

HEX byte: 0001 0101 ---> 15H
MC4 command: FI15

COMMAND: GInnnn

FUNCTION: Start executing stored program commands at line number nnnn

RANGE: nnnn = 0 to 9999

COMMENTS: The MC4 will start executing commands from line number nnnn and continue until it encounters a QP command.

nnnn can be any line number containing commands. It does not have to be the first command of a program.

If nnnn is not specified, nnnn defaults to 0000

OPTION: GInnnn-mm

- execute mm lines of commands starting with line number nnnn

EXAMPLE: GI ;start executing commands at line number 0000
GI27 ;start executing commands at line number 0027
GI30-5 ;execute 5 lines of commands starting at line number 0030

COMMAND: H (A, B, C, ... Z)

FUNCTION: Define a "jump to" label

RANGE: 26 different labels may be defined from A to Z

COMMENTS: This command marks a program location as the destination for the jump (J) command. The H and J commands enable looping for repetitive execution of the same commands.

To use this command in real-time mode, it must be enclosed in arrowhead brackets along with the other commands. See examples in the J command description section.

To use it as part of a stored program, it may be enclosed in arrowhead brackets or it can be on a separate line. See examples in the J command description section.

EXAMPLE: HA ;defined this location as A
 HS ;defined this location as S

COMMAND: Innnn

FUNCTION: Insert a line of command(s) at line number nnnn

RANGE: nnnn = 0 to 9999

COMMENTS: This command is used to edit your existing stored programs without having to re-type all the commands of that program.

After you issue this command, the MC4 will then wait for your next command and store it at line number nnnn when received.

All commands that were previously below line number nnnn, will be shifted one line number lower to make room for the new command.

If nnnn is greater than the number of existing program lines, the MC4 will place your command as the last command of the last program.

After you perform this operation, it is wise to do ask for a listing of the programs to verify that the MC4 did it correctly.

EXAMPLE: I20 ;insert the next line of command(s) at line # 20
 MW;MX ;these two commands will now be stored at line # 20

COMMAND: J (A, B, C, ... Z) nn

FUNCTION: Jump to specified label nn number of times and continue to execute commands from that location.

RANGE: nn = 0 to 99

COMMENTS: If nn = 0 or is not specified, the MC4 will loop indefinitely. To exit this endless loop, you can either press the STOP switch on the front panel or turn the MC4 off.

The same label may be used in the same program as long as they are not nested.

The number of times that a set of commands inside a loop will be executed is nn+1

EXAMPLE: Real-Time Mode:

1. <HA;MW;MX;JA5;>

This is the only construct allowed in real-time mode. The above command will move W and X at the same time 6 times. However, if you want to move W and then when W done, move X, then the command would be: Assume that NW1000 was previously sent.

2. <HA;MW;BW999;MX;JA5;>

3. <HD;MW;S1;C1;JD7;>

- start moving W and at the same time, pulse output number 1
- repeat 7 times for a total of 8 executions

4. <HD;MW;BW999;S1;C1;JD7;>

- start moving W, when W done, pulse output number 1
- repeat 7 times for a total of 8 executions

5. <HT;MW;UT100;MX;JT;>

- start moving W and executing the time delay at the same time; the time delay is for 1 second; 1 second after W started moving, move X
- repeat the process indefinitely

6. <HT;MW;BW999;UT100;MX;JT;>

- start moving W; 1 second after W is done, move X
- repeat indefinitely

Program Mode:

We will now show how the above examples 1-6 are done in the program mode.

1. <HA;MW;MX;JA5;>

2. HA;MW;MX;JA5

3. <HD;MW;S1;C1;JD7;>

4. HD;MW;S1;C1;JD7
5. <HT;MW;UT100;MX;JT;>
6. HT;MW;UT100;MX;JT

COMMAND: J(A,B,C...Z)U(1,2,3)(H,L)

FUNCTION: Jump to the specified label (U)ntil the designated bit goes to the programmed level: (H)igh or (L)ow

COMMENTS: This command enables the length of execution of a motion command or a program to be controlled by an external input signal.

EXAMPLE: NX1 Set number of steps equal to 1
 HA Define a "jump to" label
 MX Move X-axis
 JAU1L Jump to label "A" until bit #1 goes low
 .
 .
 .

COMMAND: K (W, X, Y, Z, A) E / D

FUNCTION: Enable or disable holding torque to the motor

COMMENTS: This command can be used to disable current flow to the motor when it is stationary. With no current flow, the motor will not dissipate any heat. This may be essential in a vacuum system.

If you disable holding torque while the motor is stationary, you must enable it before you make the next move.

EXAMPLE: KWE ;enable holding torque on W axis
 KXD ;disable holding torque on X axis -- no current -- no heat

COMMAND: Lnnnn

FUNCTION: Prepare a listing of currently stored programs from line number nnnn and store it in the output buffer

RANGE: nnnn = 0 to 9999

OPTIONS: Lnnnn ;list commands at line number nnnn

 Lnnnn-mmmm ;list line numbers from nnnn to mmmm

 Lnnnn- ;list line numbers from nnnn to the end

 L-nnnn ;list line numbers 0000 to nnnn

COMMENTS: For IEEE-488 interface,

- you must issue a read statement to the interface to place the MC4 into the Talker mode. At that time the MC4 will send the contents of its output buffer to your computer.
- the MC4 sends one line of commands at a time, followed by a carriage return/line feed combination regardless of the setting of switch S3-4 which only tells the MC4 how its commands are terminated. Your interface should be programmed to terminate the read operation when it receives the carriage return/line feed. Since a listing will normally be more than one line, you will have to put your read statement in a program loop so that it is continually executed until the MC4 has sent all line listings.
- After the MC4 has sent the last line of the program listing, it will then send the ASCII characters END in response to the next read statement. Your application program will use this flag to exit the read loop.

For RS-232-C interface,

- the MC4 will automatically send the listing to your computer in response to the L command.
- if the program listing is coming too fast for you computer to handle, you can cause the MC4 to stop until your computer catches up:
 - Hardware Handshake: Set Data Terminal Ready false
 - Software Handshake: Send XOFF, then send XON when ready

EXAMPLE: L ;list all programs

 L10 ;list line number 10

 L5-12 ;list lines 5 through 12

 L-20 ;list lines 0 through 20

 L30- ;list lines 30 to the end

COMMAND: M (W, X, Y, Z)

FUNCTION: Start move on designated axis

COMMENTS: After you have set up your movement parameters (i.e. programmed the number of steps with the N command, set the stepping rate with the R or T command, and defined the direction with the + or - command) you issue the M command to perform the move.

EXAMPLE: MW ;start moving W axis
MZ ;start moving Z axis

COMMAND: N (W, X, Y, Z) nnnnnnn.d

FUNCTION: Set the number of steps to be executed on the next move for the designated axis

RANGE: nnnnnnn = 1 to 7999999 ; d = 0 to 9

COMMENTS: In addition to the comments below, it is strongly advised that you refer to Chapter 10: MC4 Theory of Operation at this time to fully understand the N command.

The parameter nnnnnnn specifies the total number of full step you desire for the next move.

The parameter d specifies the fraction of a full step you desire in addition to the number of full steps.

It is possible to specify move lengths of less than one full step i.e. nnnnnnn = 0 and d = fraction of full step desired

If d = 0, it does not have to be specified.

The parameter d is not valid for the UE30 motor type. This motor cannot be moved a fraction of a full step. If you specify a fraction of a full step, the MC4 will signal an error and send the following error message: E03 PARAMETER/MOTOR ERROR

If the number of steps from move to move is the same, you do not have to re-program the N command. You only have to re-program the N command when the number of steps for the next move is different.

EXAMPLE: NW2000 ;program W axis for 2000 full steps
NX3000.7 ;program X axis for 3000.7 full steps
NY.4 ;program Y axis for 0.4 full steps

COMMAND: O (W, X, Y, Z)

FUNCTION: Initiate an origin search on the designated axis

COMMENTS: The MC4 will search for and locate the origin point defined by its axis configuration switches. Refer to Chapter 2 for the complete discussion of the axis configuration switches.

The three origin point options are:

1. floating origin defined by location where the front panel display equals zero.
2. switch mounted on a positioning stage
3. switch mounted on a positioning stage in conjunction with a top zero pulse

The MC4 cannot perform origin search on more than one axis at a given time. If this is an inconvenience for you, a possible alternative is described below in the P command section.

EXAMPLE: OW ;initiate origin search on W axis
OY ;initiate origin search on Y axis

COMMAND: P (W, X, Y, Z, A) nnnnnnn.d

FUNCTION: Move to the position specified by nnnnnnn.d relative to a predefined reference point

RANGE: nnnnnnn = -7999999 to 7999999 ; d = 0 to 9

COMMENTS: In addition to the comments below, it is strongly advised that you refer to Chapter 10: MC4 Theory of Operation at this time to fully understand the P command.

The P command is used to tell the MC4 your desired final position relative to a predefined reference point. The reference point is defined by the A command and by the reset switch on the front panel.

The P command is referred to as the absolute positioning command. It alone performs all three commands used for relative positioning.

- by comparing the present (pre-move) position and the specified parameter nnnnnnn.d it knows the direction of movement (+ or - command) and the number of steps (N command).

- when the above is calculated, it initiates the move (M command)

The parameter d is not valid for the UE30 motor type. This motor cannot be moved a fraction of a full step. If you specify a fraction of a full step, the MC4 will signal an error and send the following error message: E03 PARAMETER/MOTOR ERROR.

IMPORTANT If your UE7x stepping motor has a 200 point encoder and you have selected the encoder to drive the front panel display, you will experience the following peculiarity if you issue P commands that specify a fraction of a full step.

The P command uses the front panel information to read the present position so that it can calculate how many steps it must execute to reach the target position. Since a 200 point encoder cannot show fractions of a full step, the MC4 will assume that the fraction of a full step is zero, i.e. the present position is on a full step boundary. However, that will not be the case if you previously issued a P command with a fraction of a full step. This will cause a slight calculation error, but the final position will always be within one full step.

EXAMPLE: If you send PW100.7, the counter may read the final position as 100 or 101, depending on the alignment of the encoder. If next you send PW200.3, the MC4 will calculate a displacement of 100.3 instead of 99.6. The final position on the display will be 201.

If this is a inconvenience, you may switch the encoder count off and let the display be driven by the internal pulses. The internal pulses simulate a 2000 point encoder and thus all MC4 calculations will be correct.

The PA command will move all axes that have a motor connected to them. Any axis that does not have a motor connected to it (indicated by the illumination of both the +LIMIT and -LIMIT light bars) will be bypassed.

The MC4 cannot perform origin search on more than one axis at a time. This may be an inconvenience if you wish to do origin search for several stages and they are far from the origin point. A handy solution to this dilemma is to use the PA command as follows:

- at the beginning of your application, perform origin search sequentially on each axis.

- issue the AA command to set the present position as the reference point for the P command
- when you wish to do origin search on all axes, issue the PA0 command
- when the PA command is done, issue the origin search command to each axis. Even though your system will be at the origin point when PA0 is complete, you should perform origin search with the O command. The execution of origin search is such as to compensate for the backlash of the lead screw of the positioning stage.

EXAMPLE: PX3000 ;go to position +3000 relative to zero on the X axis

PY-1000 ;go to position -1000 relative to zero on the Y axis

PA200 ;move all stages/motors connected to the MC4/MD4 to position 200

COMMAND: QE

FUNCTION: Exit program entry mode

COMMENTS: This command is used when you are finished entering stored program commands. It returns the MC4 back to real-time mode.

As you will recall, you can enter several commands on the same line number. However, the QE command must stand alone.

COMMAND: QP

FUNCTION: Define the end of a program of commands

COMMENTS: This command signals the MC4 that it has reached the end of the program of commands it was executing. At this time, it will exit program execution mode and enter real-time mode.

Like the QE command, this command must be alone on a program line.

COMMAND: R (W, X, Y, Z, A) nnnn

FUNCTION: Set the stepping rate (steps / second) for the designated axis

RANGE: nnnn = 32 to 4000

COMMENTS: The stepping rate nnnn is defined in full steps/second. However, the MC4 pulse output rate will be 10 times nnnn. Refer to Chapter 10: MC4 Theory of Operation.

After you have programmed the stepping rate, you initiate the move at that speed by issuing the M or P command.

EXAMPLE: RW1000 ;set stepping rate at 1000 full steps/sec. on W axis

RX3000 ;set stepping rate at 3000 full steps/sec. on X axis

COMMAND: S (1, 2, 3, 4, 5, 6, 7, 8)

FUNCTION: Set designated output bit high

RANGE: There are 8 output bits which can be set high and set low

COMMENTS: The output bits are available on the MC4 General Purpose Input/Output (GPIO) connector. Refer to chapter 9 for the pin-outs. These outputs are open-collector, so you must place a pull-up resistor (value: 1K ohm to 10K ohm) between a power supply and the output pin. For the power supply, you may use the +5 volts or the +12 volts available on the GPIO connector.

EXAMPLE: S1 ;set output bit #1 high
<S3;S5;S7;> ;set output bits #3,5,7 high

COMMAND: T (W, X, Y, Z, A) nnnn

FUNCTION: Set the stepping rate by defining the time delay between full steps, i.e., program the full step period

RANGE: nnnn = 2 to 9999

COMMENTS: period = nnnn times 10 milliseconds

This command is similar to the R command because they both set the stepping rate. However, the R command parameter defines the stepping rate as a step pulse frequency, while the T command defines the stepping rate by specifying the full step period.

After you have programmed the stepping rate, you initiate the move at that speed by issuing the M or P command.

EXAMPLE: TX500 ;set stepping rate at one full step every 5 seconds on
 ;the X axis

TY9999 ;set stepping rate at one full step every 99.99 seconds
 ;on the Y axis

COMMAND: UE (1, 2, 3) H / L

FUNCTION: Wait for the designated input pin to be in the specified logic level before proceeding to execute the next command.

COMMENTS: The 3 external inputs are connected to the MC4 through the General Purpose I/O connector.

This command may be used in the MC4 interdependent mode of operation whereby you specify the command(s) to be executed when the logic state becomes true and enclose them all in arrowhead brackets.

However, you can also use this command to simply "freeze" your interface. This is done by sending the command all by itself. When the MC4 is waiting for the logic level programmed, it outputs the following signals to the interface:

IEEE-488

- not ready for data (NRFD true)

RS-232-C

- hardware handshake: clear to send false
- software handshake: it sent an XOFF and will not send an XON

The freezing of the interface is done by sending a dummy command (like C1, S2, etc.) to the MC4 while it is executing the UE command. Your interface and consequently your application program will "hang" until the programmed logic state is attained. At that time, the MC4 will accept and execute the dummy command, but more importantly, your application program will proceed to execute the next statements.

As you can see, the UE command can be used to synchronize the execution of your application program with the logic levels of up to three external signals.

The logic state of these three bits may be read with a request for status command.

EXAMPLE: <MW;UE1L;MX;>

- start W axis move; wait for input #1 to go low, then start moving on X axis

<MX;UE3H;S1;>

- start X axis move; wait for input #3 to go high, then set output #1 high

UE2L ;"freeze" interface until input #2 goes low

S1 ;then accept and execute S1

TAKE MEASUREMENT ;then application program is enabled to proceed to perform the next task like take a measurement, enable a device, etc.

COMMAND: UTnnnn

FUNCTION: Wait the specified time delay before executing the next command

RANGE: nnnn = 1 to 9999

COMMENTS: Delay = nnnn times 1 millisecond

Similarly to the UE command above, this command may be used in the MC4 interdependent mode of operation and also to "freeze" your interface. Refer to the UE command discussion on these subjects.

EXAMPLE: <MW;UT100;MX;>

- start W axis move; 1 second after W started, start X axis move

NW1000

<MW;BW999;UT200;MX;>

- start W axis move; 2 seconds after W is done, start X axis move

NW1000

MW;UT200;MX

- this is how the above example is done when entered as part of a stored program

UT500 ;"freeze" interface for 5 seconds

S1 ;then accept and execute S1

TAKE MEASUREMENT ;the application program is enabled to proceed to perform the next task like take a measurement, enable a device, etc.

COMMAND: V (0, 1, 2, 3, 4, 5, 6, 7)

FUNCTION: Define an MC4 command parameter as a variable

RANGE: 8 different variables may be defined

COMMENTS: The MC4 allows you to define variable parameters for some of its commands. This enables you program repetitive tasks with different parameters in a few lines of program code. The procedure for use of this command is as follows:

1. Define and initialize the variable. The format for variable definition is:

$V_m = nnnn$ $m = 0, 1, 2, 3, 4, 5, 6, \text{ or } 7$, and $nnnn = \text{initial parameter value}$

2. The variable can then be incremented or decremented. The format is:

$V_m = V_m + pppp$ $m = \text{variable number}$, and $pppp = \text{increment/decrement value}$

3. When the variable has been assigned a new value, it is used as a parameter for an MC4 command. See examples below.

The following commands allow variable definition as the numerical parameter:

B :wait for specified number of steps to be completed
J ;jump to specified label "n" times
N :set number of step pulses
P :set number of absolute position
R ;set step pulse rate

This command is best used as part of a stored program. When used in a stored program, all the above commands are valid. However, only the B and J commands are valid in real-time mode of operation.

EXAMPLE: **Real-Time Mode or Program Mode**

NW1000
NY10
< V0=100;MW;HA;BWV0;MY;V0=V0+100;JA9;>

- set number of steps for W axis to 1000

- set number of steps for Y axis to 10
- initialize variable to 100 and start moving W axis
- when W reaches 100 steps, move Y axis 10 steps
- increment variable to 200
- jump back, wait for W to reach 200 steps, then move Y axis again 10 steps, etc.

<V0=50;MW;HA;BWV0;S2;C2;V0=V0+50;JA19;>

- pulse output bit #2 every 50 steps of the W axis move.

Program Mode:

```
NW2000
V1=0           ;increase the step rate by 200 steps/second
HB             ;on each successive move
V1=V1+200
RWV1
MW
JB4
V3=10000
HC
PYV3           ;decrement the absolute position by 1000
UT50           ;for each successive move
V3=V3-1000
JC9
```

COMMAND: + (W, X, Y, Z)

FUNCTION: Set the direction for the next relative move to plus

EXAMPLE: +W
 +Z

COMMAND: - (W, X, Y, Z)

FUNCTION: Set the direction for the next relative move to minus

EXAMPLE: -X
 -Y

COMMAND: #nnnn

FUNCTION: Delete stored program lines

RANGE: nnnn = 0 to 9999

COMMENTS: This command is used to edit stored programs. It allows you to delete a single line of commands or several lines.

OPTIONS: #nnnn ;delete line number nnnn only

 #nnnn-mmmm ;delete lines nnnn through mmmm

 #nnnn- ;delete lines nnnn to the end

 #-nnnn ;delete lines 0000 through nnnn

EXAMPLE: #10 ;delete line number 10

 #20-30 ;delete lines 20 through 30

 #50- ;delete lines 50 to the end

COMMAND: ?

FUNCTION: Obtain MC4 status and store it into the MC4 output buffer

COMMENTS: Before issuing this command, you will use the FS command to tell the MC4 which status option you desire.

The following comments are interface specific:

IEEE-488

- you can only issue this command when the MC4 is idle, i.e., not executing any command.
- after sending this command, you must issue a read statement to the interface that will place the MC4 in the Talker mode. At that time, the MC4 will send the contents of its output buffer.
- if you want a status report when the MC4 is executing a command, you do not send the ? command. You simply issue a read statement to the interface to place the MC4 in the Talker mode. The MC4 assumes you want status.
- The response string is always terminated by a carriage return/line feed combination regardless of the setting of S3-4 which only tells the MC4 how its commands are terminated.

RS-232-C

- you issue this command when the MC4 is idle and when it is executing a command.
- the MC4 will automatically send the status information to your computer in response to the ? command.
- the MC4 will send the status information only if it sense that your computer is ready to receive.
 - hardware handshake: set Data Terminal Ready true
 - software handshake: if you previously sent XOFF, send XON

EXAMPLE: For examples and the status response format see section 6.4

CHAPTER 8: MC4 GENERAL PURPOSE INPUT/OUTPUT CONNECTOR

The functions provided on the input pins of this connector are:

1. duplication of every switch of the MC4 front panel. By grounding an input pin, the same result is obtained as if you pressed a switch on the front panel of the same designation. This feature may be used when you want front panel control but the front panel is not within reach.
2. the external inputs for the UE command. As you will recall, this command tests the logic level (high or low) of an external input. Based on that information, it will proceed or wait for the programmed logic level.

The functions provided on the output pins of this connector are:

1. step pulse and direction for each axis.
2. 8 programmable output bits controlled by the C and S command.
3. +5 volts, +12 volts and ground.

PIN#: FUNCTION:

1	+ direction motion enable
2	- direction motion enable
3	High speed enable
4	Origin search enable
5	Axis select
6	W axis output pulses
7	W axis direction
8	X axis output pulses
9	X axis direction
10	Y axis output pulses
11	Y axis direction
12	Z axis output pulses
13	Z axis direction
14	Program Select
15	W axis display reset
16	X axis display reset
17	Y axis display reset
18	Z axis display reset

PIN#: FUNCTION:

19	B1 (programmable output bits)
20	B2 controlled by C2/S2 cmd.
21	B3 controlled by C3/S3 cmd.
22	B4 controlled by C4/S4 cmd.
23	B5 controlled by C5/S5 cmd.
24	B6 controlled by C6/S6 cmd.
25	B7 controlled by C7/S7 cmd.
26	B8 controlled by C8/S8 cmd.
27	E1 External input for UE1x cmd.
28	E2 External input for UE2x cmd.
29	E3 External input for UE3x cmd.
30	Emergency stop
31	Program start
32	Program pause
33	Local/Remote
34	+12V (50 mA)
35	+5V (100 mA)
36	Logic ground
37	System reset

Direction Levels: High = positive direction Low = negative direction

All inputs are internally pulled up to +5 volts.

All outputs are open collector to 30 volts. To use an output, you must provide a pull-up resistor (1K ohms to 10 K ohms) between the output pin and a voltage supply.

CHAPTER 9: FRONT PANEL POSITION DISPLAY

The front panel position display is in units of a full step of a stepping motor. A full step is the distance between two adjacent stator poles. A ministeep is a fraction of a full step. The MC4/MD4 system is a ministeep system for permanent magnet motors and a full step system for variable reluctance motors. The UE7x series motors are permanent magnet and the UE30 motor is variable reluctance.

For the permanent magnet motors, the resolution is one-tenth of a full step, i.e., a full step is divided into 10 equal increments. If your display is being driven by a ministeep encoder (i.e., 2000 points/revolution) or you selected the no encoder option (refer to Chapter 2) the display will change by .1 for each single step actuation from the front panel. Also the smallest displacement you can program is, for example, NW.1. If you issue a MW command after sending the NW.1 command, the display will change by .1.

If your display is being driven by a full step encoder, the decimal point will not be present. The display will change when the motor has moved a full step, i.e., 10 ministeeps. This means that you will need to single step the MC4 10 times for the display to register one step. In programming mode, if you set NW.1 you will need to issue 10 MW commands for the display to register one step. Similarly, you can issue NW1 and a single MW command to achieve the same thing. With NW1 the MC4 will automatically generate 10 step pulses in response to MW.

For the variable reluctance motor, UE30, the resolution is a full step. Each single step actuation will change the display by 1. If the display is being driven by an encoder, the decimal point will not be present. If you do not have an encoder, then the display will show the decimal point, but the number to the right of the decimal point will always be zero. Commands like NW.1 are not allowed for UE30 motors. The MC4 will signal an error if you try this.

In all cases, to obtain the total displacement, simply multiply the front panel readout with the step size of your stage.

$$\text{DISPLACEMENT} = \text{DISPLAY} \times \text{STEP SIZE}$$

CHAPTER 10: MC4 THEORY OF OPERATION

The MC4 is basically an intelligent pulse source. When connected to a stepping motor driver, like the MD4, it becomes an intelligent stepping motor controller. Its command structure and front panel controls are tailored for stepping motors and stepping motor systems. As a result, we will refer to the pulses produced by the MC4 as step pulses for the remainder of this chapter.

The following step pulse attributes can be controlled:

1. frequency
2. quantity
3. direction
4. time

In the following discussion we will use the term "full step". A full step is the rotational distance between two stator poles of a stepping motor. Stepping motors are usually specified as n steps per revolution. Our UE30 stepping motor is 48 steps/revolution and our UE7x series stepping motors are 200 steps/revolution.

The paragraphs below that mention ministepping and moving fractions of a full step, pertain only to the UE7x series motor types. These are permanent magnet type stepping motors and thus they can be ministepped. The UE30 stepping motor is a variable reluctance type stepping motor and thus should not be ministepped. This motor should only be operated in the full step mode.

10.1 FREQUENCY

The frequency of the step pulses is 10 times the programmed rate. If you program a rate of 1500 full steps/second, either from the front panel or with the R command, the step pulse frequency will be 15,000 Hz.

The MC4 assumes that it is driving a stepping motor driver capable of ministepping a stepping motor. In particular, it assumes that the stepping motor driver is set to divide each full step into 10 ministeps. As a result, each step pulse from the MC4 will correspond to a motor rotation of one-tenth of a full step.

When programming a desired stepping rate, the parameter specifies full steps per second. The parameter for the R command specifies full steps per second, and the displayed rate when using the front panel also represents full steps per second.

Since a stepping motor will move only one-tenth of a full step for each step pulse from the MC4, it is necessary for the MC4 to supply step pulse at a frequency of 10 times the programmed rate in order to achieve the desired stepping rate.

As we mentioned above, the MC4 is basically a pulse source which assumes that it is supplying step pulses to a stepping motor driver that it set to rotate a stepping motor one-tenth of a full step for each pulse received. However, the stepping motor driver does not have to be set for one-tenth step division. If it is set for another step division, some useful effects may be achieved.

Our MD4 stepping motor driver can be set to divide a full step such that each step pulse received will cause a rotation of:

1. one-tenth of a full step
2. one-fifth of a full step
3. one-half of a full step
4. one full step

With the R command, we can program a stepping rate range of 32 to 4000 full steps/second. This corresponds to a step pulse frequency of 320 Hz to 40,000 Hz. The following table shows the resultant full step rate for each possible MD4 setting:

MD4 Setting	Resultant Step Rate Range (full steps/second)
1/10	32 - 4000
1/5	64 - 8000
1/2	160 - 20,000
1	320 - 40,000

For the UE30 motor type, the above discussion is valid except that the MD4 will not ministe the motor. Instead, for every 10 step pulses it receives, it moves the motor one full step.

10.2 QUANTITY

The number of step pulses outputted is 10 times the programmed number. If you program 1000 steps for the W axis by issuing the NW1000 command, the MC4 will output 10,000 step pulses when you initiate the move.

The MC4 assumes that it is driving a stepping motor driver capable of ministepping a stepping motor. In particular, it assumes that the stepping motor driver is set to divide each full step into 10 ministeps. As a result, each step pulse from the MC4 will correspond to a motor rotation of one-tenth of a full step.

The parameter for the N command specifies the number of full steps and the fraction of a full step desired for the next move. The parameter for the P command specifies the desired final position in terms of the number of full steps and the fraction of a full step.

Since a stepping motor will move only one-tenth of a full step for each step pulse from the MC4, it is necessary for the MC4 to supply a quantity of step pulses that is 10 times the programmed number in order to reach the desired position.

For the UE30 motor type, the above discussion is valid except that the MD4 will not ministepped the motor. Instead, for every 10 step pulses it receives, it moves the motor one full step.

10.3 DIRECTION

The step pulse direction controls the rotational direction of the motor. The MC4 has two channels for pulse outputs for each axis. One channel supplies positive direction step pulses which create a counterclockwise rotation of the motor. The other channel supplies negative direction step pulses which create clockwise rotation of the motor.

10.4 TIME

The MC4 has several timing options to control when it will begin to output step pulses. These are:

1. when it receives a real-time move command such as the M command and the P command and when it encounters these same commands in a stored program
2. when another axis has completed a pre-defined number of steps
3. when a pre-programmed time interval has expired
4. when an external event triggers it

CHAPTER 11: SAMPLE PROGRAMS

In this chapter we present the sample programs that you will use in Chapters 4 and 5 to establish correct communication between your host computer and the MC4.

We show one sample program written for three different IEEE-488 interfaces and an RS-232-C program for the IBM-PC and compatibles.

The three IEEE-488 interfaces are:

1. Real Time Devices for the IBM-PC
2. National Instruments for the IBM-PC
3. HP-85 *PAGE 11-6*

The RS-232-C program has two versions. One is for the hardware handshake mode and the other is for the software handshake mode.

11.1 IEEE-488 SAMPLE PROGRAMS

11.1.1 REAL TIME DEVICES

```
1 ' DIRECTIONS: 1. Set MC4 address to 6
2 '             2. Set MC4 command terminator to Carriage Return/Line Feed
3 ' *****
4 ' Lines 10-190 must be included in front of your application
5 ' program. These lines are copied from the GPINIT.BAS program
6 ' supplied on the distribution diskette by Real Time Devices,
7 ' (C) COPYRIGHT 1985
8 ' *****
9 '
10 DEFINT A-Z
11 DEF SEG = 0 : LIBSEG = VAL("&H" + HEX$(PEEK(1018) + PEEK(1019) * 256))
12 FOR LIB = 1 TO 1
13     DEF SEG = LIBSEG
14     ID = PEEK(262) + PEEK(263) * 256 : LENGTH = PEEK(ID)
15     IDS = ""
16     FOR I = 1 TO LENGTH
17         IDS = IDS + CHR$(PEEK(ID + I))
18     NEXT I
19     IF IDS = "GP100" THEN GP100 = LIBSEG
20     LIBSEG = VAL("&H" + HEX$(PEEK(254) + PEEK(255) * 256))
21 NEXT LIB
22 IF GP100 = 0 THEN PRINT "Missing GP100 Subroutine Library" : END
```



```

140 ' Initialize offsets
150 INIT1 = 256
160 CALL INIT1(INIT2,INIT3,INIT4,INIT5,VERIFY,VERSION,GPRESET,GPNEW,
    GPTIMER,PULSE,HOLD,RTL,RTLHOLD,TONLY,LONLY,EVENT,TRAP,
    ADDRESS,BUS,MESSAGE,TEXT)
170 CALL INIT2(WBYTE,RBYTE,WWORD,RWORD,WDWORD,WQWORD,RQWORD,WSTR,
    RSTR,WARRAY,RARRAY,BWRITE,BREAD,WDMA,RDMA,DMA,TERM,NOTERM,
    CRLF,EOI,PARSER)
180 CALL INIT3(CONTROL,IFC,REN,RENCLR,RENLOC,REMOTE,TCS,TCA,STANDBY,LLO,
    DCL,PPU,SPE,SPD,GTL,LOCAL,SDC,GPCLEAR,GPGET,TRIGGER)
190 CALL INIT4(UNLISTEN,UNTALK,MYLISTEN,MYTALK,LISTEN,XLISTEN,ALISTEN,TALK,
    XTALK,BUSCOM,PASS,SPOLL,APOLL,REQUEST,PPOLL,CONFIGURE,
    RESPONSE,MYSTATUS)
230 '
240 '*****
250 '
260 CALL GPRESET(STATUS)
270 CALL REN(STATUS)
280 CALL IFC(STATUS)
290 COUNT=50
300 CALL GPTIMER(COUNT)
310 CALL NOTERM
320 TRS=CHRS(10):SELECT=-1:CALL PARSER(SELECT,TRS)
330 CALL UNTALK(STATUS)
340 CALL UNLISTEN(STATUS)
350 LISSEN = 6
360 TALKER = 6
370 '
380 '*****
390 '
400 'NOTE: The MC4 command (CMMDS) can be obtained directly
410 '      from the keyboard or read from a data file
420 '
430 INPUT CMMDS                                'or read data file
435 IF CMMDS="Q" THEN GOSUB 590:GOTO 430        'read MC4 status on the fly
440 CMMDS=CMMDS+CHRS(13)+CHRS(10)
450 CALL UNLISTEN(STATUS)
460 CALL LISTEN(LISSEN,STATUS):CALL MYTALK(STATUS)
470 CALL WSTR(CMMDS,BYTES,STATUS)
480 DS=LEFTS(CMMDS,1)
490 IF DS="D" THEN GOSUB 590
500 IF DS="L" THEN GOSUB 750
510 IF DS="?" THEN GOSUB 590
520 GOTO 430
530 '
540 '*****
550 '
560 ' This routine reads the MC4 output in response
570 ' to a DISPLAY and STATUS command
580 '
590 CALL UNLISTEN(STATUS):CALL UNTALK(STATUS)
600 CALL MYLISTEN(STATUS)
620 CALL TALK(TALKER,STATUS)
630 QS=STRINGS(100," ")

```

```

640 CALL RSTR(QS,BYTES,STATUS)
650 CALL UNTALK(STATUS)
660 PS=LEFTS(QS,BYTES)
670 GOSUB 900
680 RETURN
690 '
700 '*****
710 '
720 ' This routine reads the MC4 output
730 ' in response to a LIST command
740 '
750 FOR I=1 TO 100:NEXT I          'Delay for MC4 to prepare listing
760 CALL UNLISTEN(STATUS)
770 CALL UNTALK(STATUS)
780 CALL MYLISTEN(STATUS)
790 CALL TALK(TALKER,STATUS)
800 QS=STRINGS(100," ")
810 CALL RSTR(QS,BYTES,STATUS)
820 CALL UNTALK(STATUS)
830 PS=LEFTS(QS,BYTES)
840 PRINT PS;
850 IF PS<>"END"+CHR$(13) THEN 760
860 RETURN
900 NB=1
905 LSTRG=LEN(PS)
910 IF NB>LSTRG-1 THEN 990
915 AS=MIDS(PS,NB,2)
920 HEADS=LEFTS(AS,1)
925 IF HEADS="0" THEN GOSUB 1000:NB=NB+2:GOTO 910
930 IF HEADS="1" THEN GOSUB 1100:NB=NB+2:GOTO 910
940 IF HEADS="2" THEN GOSUB 1200:NB=NB+2:GOTO 910
950 IF HEADS="3" THEN GOSUB 1300:NB=NB+2:GOTO 910
960 IF HEADS="4" THEN GOSUB 1400:NB=NB+2:GOTO 910
965 IF HEADS="W" OR HEADS="X" OR HEADS="Y" OR HEADS="Z" OR HEADS="E" THEN
    GOSUB 1500:GOTO 990
970 PRINT"":PRINT"STATUS BYTE INVALID"
990 RETURN
1000 PRINT"":PRINT"STATUS BYTE --> MOVING/STATIONARY":PRINT""
1005 STATS=RIGHTS(AS,1)
1007 S=ASC(STATS)
1010 B=S AND &HF          'clear upper nibble -- save status
1020 C=B AND 1:IF C=1 THEN PRINT"W-AXIS IS MOVING" ELSE PRINT"W-AXIS IS
    STATIONARY"
1030 C=B AND 2:IF C=2 THEN PRINT"X-AXIS IS MOVING" ELSE PRINT"X-AXIS IS
    STATIONARY"
1040 C=B AND 4:IF C=4 THEN PRINT"Y-AXIS IS MOVING" ELSE PRINT"Y-AXIS IS
    STATIONARY"
1050 C=B AND 8:IF C=8 THEN PRINT"Z-AXIS IS MOVING" ELSE PRINT"Z-AXIS IS
    STATIONARY"
1060 RETURN
1070 '
1075 '*****
1080 '
1100 PRINT"":PRINT"STATUS BYTE --> TRAVEL LIMIT WX":PRINT""

```

```

1105 STATS=RIGHT$(AS,1)
1107 S=ASC(STAT$)
1110 B=S AND &HF          'clear upper nibble -- save status
1120 C=B AND 1:IF C=1 THEN PRINT"+W LIMIT TRUE" ELSE PRINT "+W LIMIT FALSE"
1130 C=B AND 2:IF C=2 THEN PRINT"-W LIMIT TRUE" ELSE PRINT "-W LIMIT FALSE"
1140 C=B AND 4:IF C=4 THEN PRINT"+X LIMIT TRUE" ELSE PRINT "+X LIMIT FALSE"
1150 C=B AND 8:IF C=8 THEN PRINT"-X LIMIT TRUE" ELSE PRINT "-X LIMIT FALSE"
1160 RETURN
1170 '
1175 '*****
1180 '
1200 PRINT"":PRINT"STATUS BYTE --> TRAVEL LIMIT YZ":PRINT""
1205 STATS=RIGHT$(AS,1)
1207 S=ASC(STAT$)
1210 B=S AND &HF          'clear upper nibble -- save status
1220 C=B AND 1:IF C=1 THEN PRINT"+Y LIMIT TRUE" ELSE PRINT "+Y LIMIT FALSE"
1230 C=B AND 2:IF C=2 THEN PRINT"-Y LIMIT TRUE" ELSE PRINT "-Y LIMIT FALSE"
1240 C=B AND 4:IF C=4 THEN PRINT"+Z LIMIT TRUE" ELSE PRINT "+Z LIMIT FALSE"
1250 C=B AND 8:IF C=8 THEN PRINT"-Z LIMIT TRUE" ELSE PRINT "-Z LIMIT FALSE"
1260 RETURN
1270 '
1275 '*****
1280 '
1300 PRINT"":PRINT"STATUS BYTE --> 3 BIT INPUT":PRINT""
1305 STATS=RIGHT$(AS,1)
1307 S=ASC(STAT$)
1310 B=S AND &HF          'clear upper nibble -- save status
1320 C=B AND 1:IF C=1 THEN PRINT"BIT 1 HIGH" ELSE PRINT"BIT 1 LOW"
1330 C=B AND 2:IF C=2 THEN PRINT"BIT 2 HIGH" ELSE PRINT"BIT 2 LOW"
1340 C=B AND 4:IF C=4 THEN PRINT"BIT 3 HIGH" ELSE PRINT"BIT 3 LOW"
1350 PRINT"":RETURN
1370 '
1375 '*****
1380 '
1400 PRINT"":PRINT"STATUS BYTE --> WAIT UNTIL":PRINT""
1405 STATS=RIGHT$(AS,1)
1407 S=ASC(STAT$)
1410 B=S AND &HF          'clear upper nibble -- save status
1420 C=B AND 1:IF C=1 THEN PRINT"WAITING FOR NUMBER OF STEPS"
1430 C=B AND 2:IF C=2 THEN PRINT"WAITING FOR EXTERNAL EDGE TRANSITION"
1440 C=B AND 4:IF C=4 THEN PRINT"WAITING FOR TIME OUT"
1450 RETURN
1460 '
1470 '*****
1480 '
1500 PRINT"":PRINT"STATUS BYTE --> DISPLAY READOUT":PRINT""
1510 DISPS=MID$(PS,NB,(LSTRG-NB))
1520 PRINT DISPS
1530 PRINT"":RETURN
1540 C=B AND 4:IF C=4 THEN PRINT"WAITING FOR TIME OUT"
1550 RETURN
1560 '
1570 '*****
1580 '

```


11.1.2 NATIONAL INSTRUMENTS

```
10 '
20 ' DIRECTIONS: 1. Set MC4 address to 6
30 '                2. Set MC4 command terminator to CARRIAGE RETURN/LINE FEED
40 '
50 '*****
60 '
70 ' Lines 130 to 180 must be included in front of your application
80 ' program. These lines are copied from the DECL.BAS program
90 ' supplied on the distribution diskette by National Instruments,
95 ' (C) COPYRIGHT 1984,1985
100 '
110 '*****
120 '
130     CLEAR ,59000!          ' BASIC Declarations
140     IBINIT1 = 59000!
150     IBINIT2 = IBINIT1 + 3
160     BLOAD "bib.m",IBINIT1
170     CALL IBINIT1(IBFIND,IBTRG,IBCLR,IBPCT,IBSIC,IBLOC,IBPPC,IBBNA,
180                 IBONL,IBRSC,IBSRE,IBRSV,IBPAD,IBSAD,IBIST,IBDMA,
190                 IBEOS,IBTMO,IBEOT,IBRDF,IBWRTF)
200 '
210 ' This is the beginning of the application program
220 '
230 '*****
240 '
250     BDNAMES = "MC4"
260     CALL IBFIND (BDNAMES,MC4Z)
270 '
280 ' The label -- MC4 -- is assigned to address 6 in the
290 ' configuration program: -- IBCONF --
300 '
310 '*****
320 '
330 ' Set and enable read string terminator
340 ' Note: This can also be done in config. program -- IBCONF --
350 '
360     EOSVZ=&HA              'Input string terminator is line feed
370     VZ=EOSVZ+&H400         '0400 means to terminate when receive EOS
380     CALL IBEOS (MC4Z,VZ)
390 '
392     TERMS=CHRS(13)+CHRS(10) 'MC4 command terminator is CR/LF
394 '
400 '*****
410 '
420 ' This is the beginning of the listen routine
430 '
440     INPUT CMMDS              'or read data file
450     IF CMMDS="Q" THEN GOSUB 540:GOTO 440 'read status on the fly
```

```

460 CMMDS=CMMDS+TERMS
470 CALL IBWRT (MC4%,CMMDS)
480 DS=LEFTS(CMMDS,1)
490 IF DS="D" THEN GOSUB 540
500 IF DS="?" THEN GOSUB 540
510 IF DS="L" THEN GOSUB 600
520 GOTO 440
530 '
538 ' This is the beginning of the STATUS REQUEST routine
539 '
540 PS=SPACES(255)           'Read until receive 255 bytes or terminator
550 CALL IBRD (MC4%,PS)
560 PS=LEFTS(PS,IBCNT%-2)
570 GOSUB 900
580 RETURN
600 ' This is the beginning of the LIST routine
610 '
615 FOR I=1 TO 100:NEXT I    'delay for MC4 to prepare listing
620 LSTS=SPACES(255)        'Read until receive 255 bytes or terminator
630 CALL IBRD (MC4%,LSTS)
640 LSTS=LEFTS(LSTS,IBCNT%-2)
650 PRINT LSTS
660 IF LSTS<>"END" THEN 620
670 RETURN
900 ' At this location, insert statements 900-1580 from the sample IEEE
910 ' program for the Real Time Devices IEEE-488 interface.

```

11.13 HP-85

```

10 !
20 ! DIRECTIONS:
30 !
40 !     1. Set MC4 address to 6
50 !
60 !     2. Set MC4 command terminator to carriage return/line feed
70 !
80 !
90 CLEAR 7
100 RESET 7
110 DIM CMMDS[100]
115 DIM PS[100]
120 DIM LSTS[100]
130 !
140 ! Obtain the MC4 command from the keyboard by using the
150 ! " INPUT CMMDS " command ( or read a data file )
160 !
170 INPUT CMMDS
180 IF CMMDS="Q" THEN GOSUB 300: GOTO 170
190 OUTPUT 706 USING "K";CMMDS
200 AS=CMMDS[1,1]
210 IF AS="?" THEN GOSUB 300
220 IF AS="D" THEN GOSUB 300

```

```

220 IF AS="L" THEN GOSUB 400
230 GOTO 170
240 !
250 ! Receive the MC4 response to a display command and a status
260 ! request command
270 !
300 ENTER 706; PS
310 SEND 7; UNT
320 GOSUB 900
330 GOTO 170
340 !
400 ! Receive the MC4 response to a list command
410 !
420 ENTER 706; LSTS
430 SEND 7; UNT
440 DISP LSTS
450 IF LSTS<>"END" THEN 420      !loop until receive "END"
460 RETURN
470 !
900 ! At this location insert HP-85 equivalent statements of lines
910 ! 900 - 1580 of the Real Time Devices IEEE-488 sample program

```

11.2 RS-232-C SAMPLE PROGRAMS

11.2.1 IBM-PC : HARDWARE HANDSHAKE

```

10 '
20 'DIRECTIONS: 1. SET THE FOLLOWING RS232 PARAMETERS WITH THE MC4 SWITCHES
30 '      - BAUD RATE: 1200
40 '      - EVEN PARITY
50 '      - 7 DATA BITS
60 '      - 2 STOP BITS
70 '
80 '      2. SET COMMAND TERMINATOR TO CARRIAGE RETURN
85 '
90 '      3. RUN PROGRAM
95 '
100 '*****
110 '
120 OPEN "COM1:1200,E,7,2,CS65535,DSO" AS #1
130 '
140 ' NOTE: IF A PARTICULAR COMMAND EXECUTION TIME IS LONGER
150 '      THAN 65.535 SECONDS, A "DEVICE TIMEOUT" WILL OCCUR
160 '      BECAUSE THE CLEAR TO SEND SIGNAL WILL STILL BE FALSE
170 '
180 '      IF THIS IS THE CASE, USE A SOFTWARE HANDSHAKE
190 '
200 '*****
210 '
220 KOM=0:CTSEND=0

```

```

230 '
240 '*****
250 '
260 INPUT CMMDS
270 '
280 ' Enter " ? " when you wish a status report.  You can issue the " ? "
290 ' command while the MC4 is idle and while it is executing a previous
300 ' command.
310 '
320 IF CMMDS="?" THEN GOSUB 620:GOTO 260
330 '
340 ' Enter " Q " if you hear a beep from the MC4 to read the error message
350 '
360 IF CMMDS="Q" THEN GOSUB 440:GOTO 260
370 IF KOM=1 THEN GOSUB 810
380 PRINT#1,CMMDS      'SEND COMMAND TO MC4
390 IF LEFT$(CMMDS,1)="D" OR LEFT$(CMMDS,1)="d" THEN GOSUB 440
400 IF LEFT$(CMMDS,1)="L" OR LEFT$(CMMDS,1)="l" THEN GOSUB 900
410 GOTO 260
420 '
430 '*****
440 PS=""
450 WHILE LOC(1)<>0
460 INPUT#1,PS
470 GOSUB 1000          'go decipher received string
480 WEND
490 IF PS="" THEN 450
500 RETURN
510 '
520 '*****
530 '
540 ' This routine is used to send the status request command.  We must
550 ' reconfigure the communication port so that we are able to send the
560 ' status request command even if the MC4 is outputting "CLEAR TO SEND"
570 ' false ( which is the basis of the hardware handshake ).
580 '
590 ' Note that the first status request will take some time because
600 ' opening the communication port is slow.
610 '
620 IF CTSEND=1 THEN 660
630 CLOSE 1
640 OPEN "COM1:1200,E,7,2,CSO,DS0" AS #1
650 KOM=1:CTSEND=1
660 PRINT#1,CMMDS
670 GOSUB 440
680 RETURN
690 '
700 '*****
710 '
720 ' This routine reconfigures the communication port so that we cannot
730 ' send commands to the MC4 if its is outputting " CLEAR TO SEND "
740 ' false.  This routine is executed the first time we wish to send a
750 ' a command to the MC4 after we have previously configured the port to
760 ' ignore the " CLEAR TO SEND " signal.

```



```

770 '
780 ' Note that the MC4 command which requires this routine to be excuted
790 ' will start late because opening a communication port is slow.
800 '
810 CLOSE 1
820 OPEN "COM1:1200,E,7,2,CS65535,DS0" AS #1
830 KOM=0:CTSEND=0
840 RETURN
850 '
900 '*****
905 '
910 LSTS=""
920 WHILE LOC(1)<>0
930 INPUT#1,LSTS
940 PRINT LSTS
950 WEND
960 IF LSTS="" THEN 900
970 RETURN
980 '
990 '*****
995 '
1000 ' At this location, insert statements 900-1580 from the sample IEEE
1010 ' program for the Real Time Devices IEEE-488 interface.

```

11.2.2 IBM-PC : SOFTWARE HANDSHAKE

```

10 '
20 'DIRECTIONS: 1. SET THE FOLLOWING RS232 PARAMETERS WITH THE MC4 SWITCHES
30 '      - BAUD RATE: 1200
40 '      - EVEN PARITY
50 '      - 7 DATA BITS
60 '      - 2 STOP BITS
70 '
80 '      2. SET COMMAND TERMINATOR TO CARRIAGE RETURN
85 '
90 '      3. RUN PROGRAM
95 '
100 '*****
110 '
120 OPEN "COM1:1200,E,7,2,CS0,DS0" AS #1
130 EKSON=1
140 '
150 '*****
160 '
170 INPUT"MC4 COMMAND: ";CMMDS
180 IF CMMDS="Q" THEN GOSUB 310:GOTO 170 'SEND Q TO READ ERROR IF MC4
    BEEPED
190 IF CMMDS="?" THEN 220 'IGNORE XOFF IF WANT STATUS
200 IF LOC(1)<>0 THEN GOSUB 440
210 IF EKSON<>1 THEN 200
220 PRINT#1,CMMDS
230 '

```

```

240 CS=LEFTS(CMMDS,1)
250 IF CS="?" THEN GOSUB 310
260 IF CS="L" OR CS="1" OR CS="D" OR CS="d" THEN GOSUB 530
270 GOTO 170
280 '
290 '*****
300 '
310 PS=""
320 BS=INPUTS(1,#1)
330 IF BS=CHRS(17) THEN EKSON=1:BS="":GOTO 320
340 IF BS=CHRS(19) THEN EKSON=0:BS="":GOTO 320
350 INPUT#1,PS
360 PS=BS+PS
370 PRINT PS
380 GOSUB 640
390 RETURN
400 '
410 '*****
420 '
430 '
440 WHILE LOC(1)<>0
450 BS=INPUTS(1,#1)
460 IF BS=CHRS(17) THEN EKSON=1
470 IF BS=CHRS(19) THEN EKSON=0
480 WEND
490 RETURN
500 '
510 '*****
520 '
530 BS=INPUTS(1,#1)
540 IF BS=CHRS(19) THEN EKSON=0
550 LSTS=""
560 INPUT#1,LSTS
570 PRINT LSTS
580 IF CS="D" OR CS="d" THEN 600
590 IF LSTS<>"END" THEN 560
600 RETURN
610 '
620 '*****
630 '
640 '
650 ' INSERT HERE STATUS INTERPRETATION ROUTINE FROM THE REAL TIME DEVICES
660 ' IEEE-488 SAMPLE PROGRAM
670 RETURN

```

CHAPTER 12: ILL-CONDITION STATUS MESSAGES: PROBLEMS AND SOLUTIONS

The MC4 is designed to detect user and interfacing errors and report back a message that will help the user quickly determine the nature of the error. The two major categories of errors are operational errors and communication errors. Operational errors are reported over the communication channel. We assume that the communication channel is functioning, but we have incorrectly programmed the MC4 or its status does not allow us to do what we want at this time. The format of the error message is the following:

E_{xx} ERROR MESSAGE

E is the header byte indicating that an error message follows

xx is the code number for the particular error

ERROR MESSAGE is a description of the type of error

Interfacing errors are reported on the MC4 front panel. If we have a communication problem, we cannot use the communication channel to receive an error message. The error is reported by a flashing number indicator in the MC4 program select window.

12.1 OPERATIONAL PROBLEMS

E01 COMMAND ERROR

First character in command string is not in the MC4 command set

E02 PARAMETER ERROR

- Second character is not a valid axis designator, or
- numerical parameter is out of allowed range

E03 PARAMETER/MOTOR ERROR

Command specifies fraction of a full-step for an axis programmed to drive a UE30 stepping motor. Fractions of a full step can only be programmed for UE7x stepping motors. Check which motor type the axis is set for by using the RESET switch of that axis and the PROGRAM SELECT switch. Refer to section 1.4.

E04 Not Used

E05 BATTERY RAM PROTECTED

The write protect switch is active. Its position will now allow writing in the battery back-up RAM.

E06 OPERATOR BUFFER FULL

Command string too long; maximum allowed length is 255 bytes.

E07 PROGRAM BUFFER FULL

Battery back-up RAM is full; capacity is 8192 bytes of compactly coded commands - not 8192 bytes of ASCII coded commands.

E08 VERIFICATION ERROR!!!

Stored program contains an error. This can be a command error or a parameter error. The error will be a result of a some mishap to the battery back-up circuitry. You will have to re-enter your program.

E09 Not Used

E0A Not Used

E0B PARITY ERROR

Parity error exists for current command. Check the MC4 word characteristic switches and your program's setting for desired parity.

E0C CHARACTER LOST

Current command overran the previous command. This error is a result of an improper handshake between the MC4 and your host computer. You should not send a command as long as the MC4 indicates it is busy. For the hardware handshake, the MC4 will indicate busy by holding the CLEAR TO SEND line false (at -12V). For the software handshake, the MC4 is busy as long as it has not sent an XON after it has sent an XOFF.

E0D FRAME ERROR

Error exists in the start and stops bits of the character sent. Check the MC4 word characteristic switches and your program's setting for the number of stop bits.

E34 ACTIVE LIMIT: + W

Command specified a W axis move in the positive direction and the W axis is at positive direction travel limit or travel limit reached during move.

E35 ACTIVE LIMIT: +X

E36 ACTIVE LIMIT: +Y

E37 ACTIVE LIMIT: +Z

E38 ACTIVE LIMIT: -W

E39 ACTIVE LIMIT: -X

E3A ACTIVE LIMIT: -Y

E3B ACTIVE LIMIT: -Z

12.2 IEEE-488 AND RS-232-C COMMUNICATION PROBLEMS

The MC4 is designed to detect some communication errors. If the implementation is incorrect, the MC4 will time-out and report what it was waiting for and never received. The MC4 will indicate the particular error by flashing a number in the program select window and beeping 10 times. The possible numbers that the MC4 may flash and their corresponding meaning is given below.

Error numbers 1 and 2 pertain only to the IEEE-488 listen function.

Error numbers 3 and 4 pertain only to the IEEE-488 talk function.

Error numbers 5 and 6 pertain only to the RS-232-C.

Flashing #: Error Indicated:

-
- | | |
|---|--|
| 1 | a. incorrect terminator programmed on MC4 dip switch
b. host computer did not send terminator
c. data bits garbled |
|---|--|
-

- | | |
|---|---|
| 2 | MC4 internal chip not initialized properly; power down and power up MC4 |
|---|---|
-

- | | |
|---|--|
| 3 | MC4 did not recognize the NDAC line transition from low to high coming from your host computer. The transition is not recognized because there was none or the opposite transition from high to low came too quickly, i.e., the NDAC pulse from your computer is too narrow. |
|---|--|
-

You will receive only the first character of the MC4 response string.

4 MC4 did not recognize your computer asserting not ready for data true.

- 5
- a. incorrect terminator programmed on MC4 dip switch
 - b. host computer did not send terminator
 - c. data bits garbled
-

6 host computer indicating it is not ready to receive data from the MC4

Hardware Handshake:

- pin 20 of RS-232-C connector at -12V. It must be at +12V to indicate ready.

Software Handshake:

- pin 20 of RS-232-C connector at -12V. It must be at +12V to indicate ready.

- OR -

- host computer has previously sent XOFF and has not sent XON
-

12.3 OTHER PROBLEMS AND POSSIBLE CAUSES

12.3.1 IEEE-488

Problem: No MC4 response to a command or request for status

Possible Cause: MC4 address incorrect

12.3.2 RS-232-C

Problem: No MC4 response to a command or request for status

- Possible Cause:
- a. connecting cable incorrect; receive data and transmit data pins reversed; ground connection missing
 - b. host computer sending data to wrong communication port
 - c. host computer not sending data

12.3.3 OPERATIONAL

Problem: MC4 executes P command improperly; final position is not the one programmed with the P command

Possible Cause: MC4 not properly programmed for the encoder type driving the display counter in error

POD 7 COPY

DO NOT REMOVE

www.manualsplus.com



1011017